Universidad Zaragoza
1542

Trabajo Fin de Grado

Detección de galerías laterales de un túnel basada en mapa previo

English title: ***Detection of sides galleries of a tunnel base on a previous map***

Autor:

Ismael Ruiz Ibáñez

Director:

José Luis Villarroel Salcedo

Grado de Ingeniería Electrónica y Automática

Escuela de Ingeniería y Arquitectura

Zaragoza, Septiembre 2019

Resumen

El método más sencillo de localización de un robot móvil es la odometría. Sin embargo, el deslizamiento de las ruedas con respecto al suelo provoca errores acumulativos que impiden el uso de la odometría como método principal de localización.

Otro de los métodos más utilizados de localización en exterior es usar el posicionamiento y localización GNSS (Global Navigation Satellite System). Sin embargo, en túneles de gran longitud, más allá de unas decenas de metros, la señal de los satélites no llega o se ve afectada por rebotes.

El problema de la perdida de la localización mediante odometría o GNSS, se puede resolver incorporando un láser. El láser proporciona información sobre el entorno del robot. Cuando el robot se encuentra en un túnel con muros lisos, con el láser se puede conocer la distancia transversal al muro y él ángulo, pero no la distancia longitudinal. Esta se puede conocer cuando se detecte con el láser algún elemento estructural que poseen los túneles, como una galería o un refugio. Al detectarse estos elementos estructurales, se elimina el problema de los muros lisos y se puede conocer la localización del robot.

Por lo tanto, el objetivo principal es la localización discreta de un robot en un mapa conocido, mediante la detección de las galerías que posee un túnel usando un sensor láser de haz plano. Para conseguir dicho objetivo se necesita tanto la información del entorno del robot, la cual se consigue con el láser, como que se conozca el mapa del túnel. Conociendo ambas, se realiza la detección y reconocimiento de las galerías, para actualizar con precisión la localización del robot, mediante el emparejamiento de los puntos del láser con un patrón de puntos obtenido del mapa conocido. Se realiza un reconocimiento de las galerías porque son características reconocibles con respecto a los muros lisos del túnel.

La metodología para conseguir el objetivo consiste en una primera implementación y puesta a punto de códigos para detectar las galerías en MATLAB y una posterior modificación e implementación de estos códigos para utilizarlos en ROS sobre un robot real o simulaciones.

Finalmente, se verifican las correctas implementaciones a través de pruebas mediante simulaciones con trazas reales del túnel del Somport o en túneles sintéticos realizados en el simulador Gazebo.

Índice

Capítulo 1

Introducción

1.1 Problema de localización en túneles

El método más sencillo de localización de un robot móvil es la odometría. Se estima el avance del robot en función del giro de las ruedas, a través de las ecuaciones e.1, e.2 y e.3. Sin embargo, el deslizamiento de las ruedas con respecto al suelo provoca errores acumulativos que impiden el uso de la odometría como método principal de localización.

$$\begin{pmatrix}\Delta x\\ \Delta y\\ \Delta\theta\end{pmatrix} = \begin{pmatrix}\Delta s * \cos\left(\theta + \frac{\Delta\theta}{2}\right)\\ \Delta s * \sin\left(\theta + \frac{\Delta\theta}{2}\right)\\ \Delta\theta\end{pmatrix} \quad \text{(e.1)}$$

$$\Delta s = \frac{\Delta s_d + \Delta s_i}{2} \approx v * \Delta t \quad \text{(e.2)}$$

$$\Delta\theta = \frac{\Delta s_d - \Delta s_i}{L} \approx w * \Delta t \quad \text{(e.3)}$$

Uno de los métodos más utilizados de localización en exterior es usar el posicionamiento y localización GNSS (*Global Navigation Satellite System*). Sin embargo, en túneles de gran longitud, más allá de unas decenas de metros, la señal de los satélites no llega o se ve afectada por rebotes.

El problema de la perdida de la localización mediante odometría o GNSS, se puede resolver incorporando un láser. El láser proporciona información sobre el entorno del robot. Aunque, esta información no siempre permite que el robot este localizado en todo momento, debido al problema de los muros lisos. Cuando el robot se encuentra en un túnel con muros lisos, con el láser se puede conocer la distancia transversal al muro, coordenada Y en la figura 1, y él ángulo, pero no la distancia longitudinal, coordenada X en la figura 1. Esto es debido a que los muros no presentan ninguna característica que permita calcular el avance en la dirección axial.

La distancia longitudinal, junto con el ángulo, se puede conocer cuando se detecte con el láser algún elemento estructural que poseen los túneles, como una galería o un refugio. Al detectarse estos elementos estructurales, se elimina el problema de los muros lisos y se puede conocer la localización del robot.

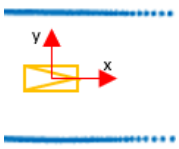

Figura 1: Ejes de coordenadas del robot

1.2 Antecedentes

Este trabajo de fin de grado se enmarca en el proyecto Navegación y Despliegue de Robots en Entornos Desafiantes, también conocido como ROBOCHALLENGE (DPI2016-76676-R), que realiza el grupo de investigación de Robótica, Percepción y Tiempo Real (ROPERT) del Instituto Universitario de Investigación en Ingeniería de Aragón (I3A).

ROBOCHALLENGE se basa en investigaciones experimentales y teóricas en ámbitos desafiantes mediante robots aéreos y terrestres. Estos entornos desafiantes, son aquellos donde, por diferentes causas, fallan o no funcionan correctamente técnicas robóticas existentes actualmente o en desarrollo. Este fracaso es debido a características del entorno, como la ausencia de propiedades visuales o geometrías discriminantes, algo que puede pasar en túneles y tuberías, entornos con sectores de diferentes características o la ausencia de una iluminación de calidad. En estos entornos, al no disponer de localización GNSS, los patrones de propagación de la señal son diferentes a los de un entorno al aire libre. Los entornos de experimentación que se plantean son estructurados, poco estructurados o nada estructurados. Este trabajo de fin de grado se basa en un entorno de experimentación estructurado, ya que se trabaja con túneles y galerías.

En este proyecto se trabaja con robots terrestres (UGVs, *unmanned ground vehicle*, que son vehículos en contacto con el suelo sin un conductor dentro) en entornos, como túneles, galerías y minas, que se pueden considerar como 2D o 2.5D, y con robots aéreos (UAV, *unmanned aerial vehicle*, que son vehículos aéreos no tripulados como drones) en entornos por donde no pueden circular los UGVs. En este caso se trabaja con robots terrestres.

A estos robots se les realiza una integración adaptativa, de sensores de barrido plano láser, como el empleado en este trabajo, para interpretar el entorno y conseguir el objetivo de tener el robot localizado, y que pueda navegar, durante toda la misión.

Este trabajo de fin de grado se enmarca dentro de la sección destinada a la localización y navegación de robots en túneles y galerías.

1.3 Objetivos

El objetivo principal es la localización discreta de un robot en un mapa conocido, mediante la detección de las galerías que posee un túnel usando un sensor láser de haz plano. Para conseguir dicho objetivo se necesita tanto la información del entorno del robot, la cual se consigue con el láser, como que se conozca el mapa del túnel. Conociendo ambas, se realiza la detección y reconocimiento de las galerías, para actualizar con precisión la localización del robot, mediante el emparejamiento de los puntos del láser con un patrón de puntos obtenido del mapa conocido. Se realiza un reconocimiento de las galerías porque son características reconocibles con respecto a los muros lisos del túnel.

La metodología para conseguir el objetivo consiste en una primera implementación y puesta a punto de códigos para detectar las galerías en MATLAB y una posterior modificación e implementación de estos códigos para utilizarlos en ROS sobre un robot real o simulaciones.

Finalmente, se verifica las correctas implementaciones a través de pruebas mediante simulaciones con trazas reales del túnel del Somport o en túneles sintéticos realizados en el simulador Gazebo.

1.4 Contenidos de la memoria

En este apartado se realiza un breve resumen de los capítulos importantes en los que se ha dividido la memoria.

- **Introducción:** El capítulo 1 se resume en explicar el problema de localización de robots en un túnel, como los antecedentes a este trabajo y los objetivos a conseguir. También se incluye un pequeño resumen de los contenidos de la memoria.
- **¿Cómo es un túnel y cómo afectan sus características a un robot?:** El capítulo dos explica porque los túneles son así y como pueden afectar en la localización de un robot las características que estos poseen.
- **Descripción del método:** El capítulo 3 plantea una descripción detallada de cada uno de los pasos que conforman el método de detección de galerías.
- **Análisis de los factores que inciden en el método:** El capítulo 4 analiza los factores que inciden en el método con la realización de estudios. También, se plantea en este capítulo la posibilidad de utilizar el método para detectar apartaderos y refugios, y obtener el patrón de puntos de una forma automática.
- **Implementación de Matlab a ROS:** En el capítulo 5 se explica que es ROS y la implementación del método para utilizarlo en dicha herramienta.
- **Pruebas:** En el capítulo 6 se agrupan todas las pruebas realizadas, en MATLAB y en ROS, para garantizar el funcionamiento de los métodos
- **Conclusiones:** El capítulo 7 contiene las conclusiones de este trabajo.
- **Bibliografía:** En el capítulo 8 se recogen todos los documentos, páginas web o libros utilizados para la realización del trabajo.

Capítulo 2

¿Cómo es un túnel y cómo afectan sus características a un robot?

Los túneles ferroviarios o carreteros de más de 500 metros son estructuras que facilitan la comunicación, y pueden jugar un papel decisivo en el funcionamiento y desarrollo de la economía de la zona.

Estas infraestructuras, como muchas otras, requieren inspecciones periódicas, para que no ocurra ninguna desgracia, reparaciones, vigilancia y, a veces, misiones de rescate.

Los accidentes, cuando ocurren en un túnel, pueden ser muy graves. Debido al confinamiento, si estos accidentes involucran incendios, pueden ocurrir consecuencias dramáticas.

Como consecuencia de los trágicos accidentes en el túnel de Mont Blanc (Francia) en 1999 y en el túnel de San Gotardo (sur de Suiza) en 2001, la Unión Europea creo la directiva 2004/54/CE del parlamento europeo y del consejo sobre requisitos mínimos de seguridad para túneles de la red transeuropea de carreteras **[1]**. Debido a esto, se identificaron un total de 515 túneles de carretera de más de 500m de longitud, llegando a alcanzar una longitud total de más de 800km de estos túneles.

Para cumplir con esta directiva, los túneles deben cumplir una serie de requisitos e integrar un conjunto de elementos estructurales para mejorar la seguridad. Algunos ejemplos de la directiva son:

- Salidas de emergencia como salidas directas desde el túnel hacia el exterior, conexiones cruzadas entre los conductos del túnel, salidas a una galería de emergencia o, también, refugios con una ruta de escape separada del conducto del túnel. La separación entre dos salidas de emergencia no debería de sobrepasar una distancia de 500m.

- En los túneles de doble cavidad, donde las cavidades están al mismo nivel o cerca, se proporcionan conexiones cruzadas al menos cada 1500m, adecuadas para el uso de servicios de emergencia.

- Se proporcionan estaciones de emergencia o apartaderos, ejemplo en la figura 2, cerca de los portales y en el interior del túnel, que para los nuevos túneles no excederá de una separación de 150m entre ellas.

Figura 2: Apartadero en un túnel

Por lo tanto, las normas de seguridad de túneles proporcionan un conjunto de características estructurales relevantes, que pueden usarse para la localización dentro de túneles. Como se ha mencionado, muchos de ellos son periódicos a lo largo de la dimensión longitudinal. Esto es una ayuda, ya que la no disposición de señal GNSS limita el número de sensores que se pueden utilizar para recibir información útil de la localización. También, las dimensiones del túnel, que son más largas que anchas, producen un crecimiento continuo de la incertidumbre de localización longitudinal, la cual se podrá reducir por la detección de galerías.

Las técnicas más comunes para localización en lugares internos son la odometría visual y SLAM (*Simultaneous Localization and Mapping*). Es obligatorio para estos algoritmos encontrar y combinar características para reducir la incertidumbre de localización. La falta de características visuales y, a veces, la oscuridad limita el uso de sensores de visión.

Como se menciona en el primer capítulo, las paredes de los túneles son uniformes en secciones largas. Por lo tanto, los sensores láser no proporcionan una información útil de la localización en la dimensión longitudinal del túnel. Sin embargo, sí que se proporciona un posicionamiento transversal preciso.

Los problemas mencionados anteriormente hacen que los entornos de túneles sean un desafío para fines de localización, que se consigue mediante la detección de galerías, refugios, etc.

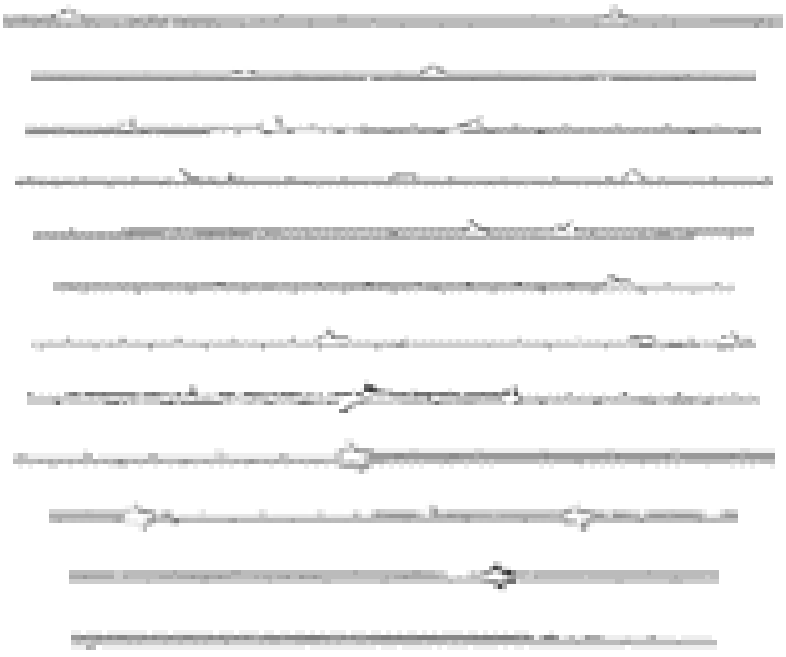

Figura 3: Túnel completo de Somport dividido en secciones

Las trazas que se utilizan en el trabajo fueron tomadas del túnel de Somport, cuyo mapa completo se encuentra en la figura 3 dividido en secciones. Este túnel tiene una longitud de 7 kilómetros y posee 17 galerías.

Figura 4: Sección con mayor detalle del túnel de Somport

En la figura 4, se muestra una sección del mismo túnel más detallada y en la que se encuentra una galería.

Capítulo 3

Descripción del método

En este capítulo se van a ir describiendo todos los puntos importantes que se encuentran en el método desarrollado para una perfecta localización y que primero se implementa en MATLAB**[2]**.

3.1 Patrón

Lo necesario para localizar al robot en el túnel es saber dónde se puede reconocer mejor y con exactitud el entorno del robot. Como se dice anteriormente, intentar localizar al robot a través de las paredes de los túneles es imposible porque estas suelen tender a ser homogéneas, y sin características diferenciales entre ellas para llegar a una detección y reconocimiento de características buena de donde exactamente nos encontramos, algo que se puede verificar en la figura 5.

Figura 5: Interior túnel ferroviario de Somport

Si te pones a reconocer una parte de la pared del túnel, el método detecta continuamente que el robot se encuentra siempre en la misma posición, aunque este se mueva, por la semejanza de ellas. En cambio, reconocer las paredes viene bien para saber la posición vertical donde se encuentra el robot respecto al túnel, algo que puede servir para evitar colisiones.

Como se ha mencionado con anterioridad, la solución más idónea para localizar al robot es detectar en su entorno las galerías que poseen los túneles. Con las galerías es posible porque cada una de ellas tiene sus propias características.

Las galerías se pueden reconocer cuando el láser detecta una distancia mucho mayor a la de las paredes, pero de esta forma solo se sabe que hay una galería, pero no se llega a identificar cuál de ellas es. Por lo tanto, esta idea es incompleta.

Otra manera de reconocerlas, la escogida finalmente, es a través de un patrón único para cada una de ellas y así diferenciarlas. Este patrón está constituido por un conjunto de puntos que recrean el contorno de cada una de las galerías. El método, con el patrón de puntos, consiste en ir comparando en tiempo real este patrón con los puntos que nos proporciona el láser hasta que estos se emparejen correctamente, lo que significa que nos encontramos en esa galería y está el robot perfectamente localizado, ya que se posee un mapa del túnel. En la figura 6, se puede ver cómo encaja un patrón con los puntos de la traza del láser.

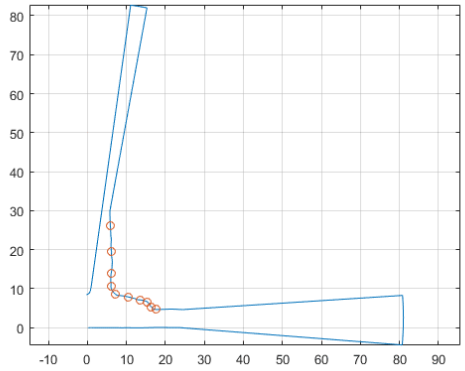

Figura 6: Patrón emparejado con los puntos de las trazas del túnel

Gracias al patrón también se puede conocer la posición relativa del robot metros antes y después del punto exacto desde el que se ha tomado el patrón de puntos de la galería con una incertidumbre reducida. Se explica que es la incertidumbre en el apartado 3.3.

El patrón de puntos se puede realizar de forma manual, parando la traza que compone el túnel en la simulación, para que los puntos clave, que forman parte del patrón, se puedan tomar de forma más precisa, o creándose el patrón automáticamente con una serie de características y filtrados que son explicados posteriormente.

La localización de cada uno de estos puntos, respecto al contorno de la galería, y la cantidad de estos se explica posteriormente en el capítulo 4.

3.2 Alineamiento con los muros

El segundo punto importante al desarrollar el método, es que el patrón se quede alineado en todo momento con los puntos del láser de la traza, para que se pueda detectar. En el caso en que el robot se desplaza continuamente en paralelo a las paredes, la traza del láser sí que se encuentra en la misma posición y ángulo que el patrón de puntos y, por lo tanto, no se realiza ningún cambio. Para este caso se consigue una óptima localización. Por el contrario, si el robot se mueve con un cierto ángulo respecto a las paredes del túnel, la traza no se encuentra alineadas con el patrón y, en consecuencia, no se obtiene una detección de la galería como en el caso anterior.

Cuando el robot no va en paralelo a las paredes del túnel, el método realiza un alineamiento del robot con los muros. Esto se consigue girando con un cierto ángulo la traza, para que los puntos de esta se encuentren alineados con el patrón.

Para que se conozca la distancia y el ángulo con el que se encuentra el robot respecto de los muros, y se sepa con qué ángulo se gira la traza, se realiza la transformada de Hough, propuesta por Paul Hough, **[3] [4]**. Con esta transformada es posible identificar una recta, en el caso presente con la traza del láser de los muros, y expresarla matemáticamente. Decir que con esta técnica también se pueden encontrar otros tipos de figuras, como circunferencias o elipses.

El primer paso a realizar consiste en eliminar los puntos de la traza que se encuentran a la derecha del robot, es decir, las que tienen la coordenada Y negativa, porque solo quiero detectar galerías a la izquierda del robot. La dirección de cada una de las coordenadas se encuentra en la figura 1. Ahora, si se representan los puntos de la traza en una imagen, solo hay puntos que

corresponden a la pared que se encuentra a la izquierda. Estos puntos recrean una única recta que queda definida por la ecuación e.4.

$$y = m * x + n \quad \text{(e.4)}$$

Es necesario pasar la ecuación a coordenadas polares, debido a que, si hay una recta vertical, no se definen los parámetros m y n de la recta.

La ecuación en coordenadas polares, e.5, donde ρ representa la distancia entre el origen de coordenadas y el punto (X, Y), y θ es el ángulo del vector director de la recta perpendicular a la original, la cual cruza el origen de coordenadas. Ambas coordenadas quedan representadas en la figura 7.

$$y = \left(\frac{-cos\theta}{sin\theta}\right) * x + \left(\frac{\rho}{sin\theta}\right) \rightarrow \rho = x * cos\theta + y * sin\theta \quad \text{(e.5)}$$

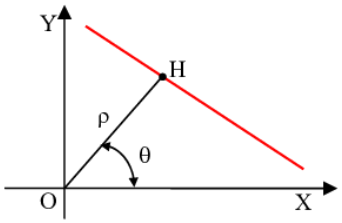

Figura 7: Variables de la transformada de Hough

Para la implementación de esta transformada es necesario crear dos vectores, que contienen los posibles valores que pueden tener ρ y θ. El primero de ellos con un intervalo de [-30,30] incrementando en 0.2. El intervalo tiene estos valores límite porque es el rango típico de los láseres que se comercializan. El segundo con un intervalo de [-π,0] con un incremento de π/180, porque los láseres suelen tener una periferia de media circunferencia. Posteriormente, se realiza la matriz de la transformada de Hough, con tantas filas como la longitud del vector ρ y tantas columnas como la longitud del vector θ. Después, se guardan todas las ρ, resultantes de realizar la ecuación e.5 con todos las θ del vector, en un vector que acumula los resultados. Luego, se realiza un histograma con el mismo número de celdas que la longitud del vector de ρ, para cada una de las columnas del acumulador. Todos los histogramas se añaden a la correspondiente columna de la matriz Hough. Después, se buscan las posiciones con los mayores valores de la matriz, y luego se busca el máximo de estos. Este valor máximo es la posición donde se encuentran los valores que determinan la recta de la imagen en los vectores de ρ y θ.

Una vez que es conocida θ, gracias a la transformada de Hough, se calcula la alfa que sirve para realizar la matriz de rotación para el láser. Alfa saldrá de la ecuación e.6.

$$\alpha = -(\frac{\pi}{2} + \theta) \quad \text{(e.6)}$$

Se realiza una rotación en el eje Z con un ángulo α de los puntos de la traza del láser a través de la siguiente matriz de rotación, e.7.

$$Rot(z,\alpha) = \begin{pmatrix} cos\alpha & -sin\alpha \\ sin\alpha & cos\alpha \end{pmatrix} \quad \text{(e.7)}$$

La matriz en la ecuación e.7 es 2x2 y no 3x3 como debería ser, porque el láser no proporciona la información del eje Z. Debido a esto, se considerado oportuno eliminar de la matriz de rotación la fila y la columna que corresponden al eje Z quedando una matriz 2x2.

Finalmente, se realiza el producto de la matriz de rotación con los puntos de la traza del túnel y estos ya están alineados con el patrón, para que se consiga una perfecta localización.

La ρ resultante de la transformada de Hough se suma a las referencias en el eje Y de todos los puntos tomados por el láser.

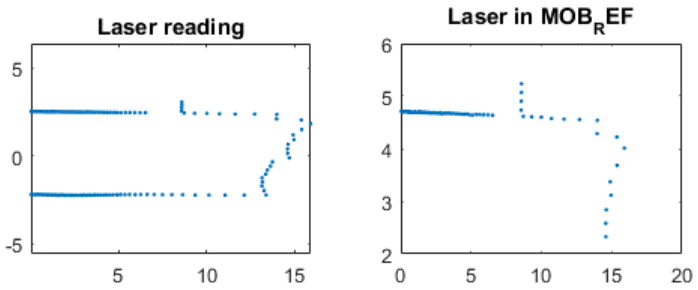

Figura 8: Puntos de la traza del láser y puntos en la referencia del robot

En la figura 8, se puede ver el cambio que se produce en las trazas al realizar todo el proceso explicado anteriormente. Es este caso apenas se han girado las trazas, debido a que el robot va prácticamente paralelo a los muros, pero en la figura 9, sí que se puede con mayor claridad el giro que se le realiza a las trazas.

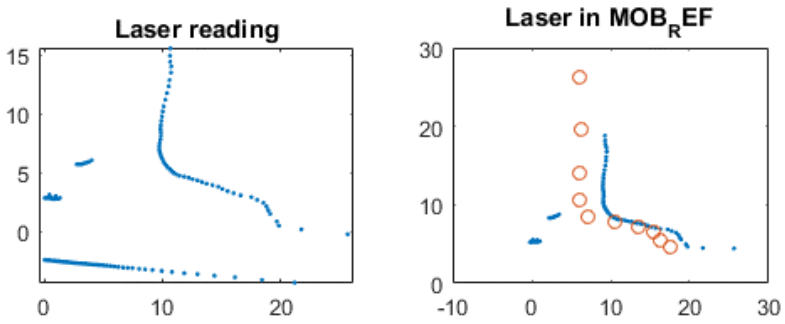

Figura 9: Traza real del láser y traza en la referencia del robot

3.3 Métricas

Una vez que el patrón está alineado con las trazas, se debe encontrar la manera de emparejar ambas. Estas se emparejan a través de diferentes métricas, que nos dan como resultado un vector con todas las distancias entre el patrón y la traza. De cada métrica nos saldrá un vector de distancias con tantos valores como puntos conformen el patrón. La figura 10 muestra cómo funciona cada métrica.

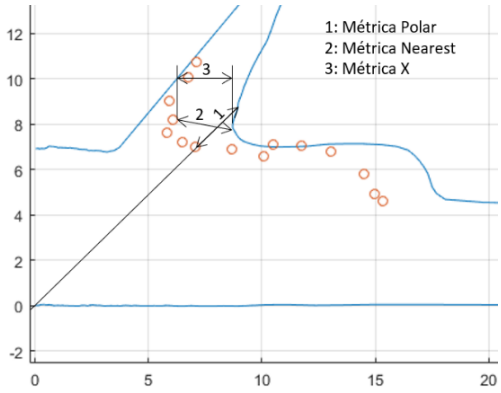

Figura 10:Funcionamiento de las diferentes métricas

La primera métrica se nombra Polar, pues en ella se trabaja con los puntos del láser y el patrón en coordenadas polares. Esta métrica consiste en comparar todas las coordenadas angulares de los puntos del láser con cada una de las coordenadas angulares de los puntos, que conforman el patrón. Si un punto de la traza tiene un valor de la coordenada angular menor a la del punto, del patrón, que se está comparando, la distancia entre ellos es el valor absoluto entre el valor de la coordenada radial del punto de la traza menos el valor de la coordenada radial del punto del patrón, y este será incluido en el vector de distancias. De modo contrario, si no se cumple esta condición en ningún momento, se añadirá al vector de distancias una distancia infinita. Como hay que darle un valor numérico, para luego compararlas, se decide poner una distancia de ochenta metros entre ambos puntos, porque era el rango del láser que se utilizó para crear la traza. Nos saldrán un vector de distancias con tantos valores como puntos conformen el patrón.

La segunda métrica, denominada Nearest, consiste en determinar la distancia más cercana entre todos los puntos del láser a cada punto del patrón. La distancia se calcula a través de la ecuación e.8, que es la distancia entre dos puntos basada en el teorema de Pitágoras, siendo la distancia la hipotenusa del triángulo rectángulo que forman ambos puntos, y se guarda la distancia menor en el vector. En MATLAB existe el comando dsearchn, que realiza esta función, también devolviendo el vector de distancias.

$$d = \sqrt{(lx - px)^2 + (ly - py)^2} \quad \text{(e.8)}$$

La última métrica es nombrada X. Esta métrica consiste en realizar una resta entre una coordenada Y de un punto del patrón, para cada una de las coordenadas Y de los puntos del láser. Si el valor mínimo de los valores resultantes es menor a 1.5m, se añade al vector el valor absoluto de la resta entre el valor de la coordenada X del punto del patrón, y el valor de la coordenada X del punto del láser con el que se obtiene el valor mínimo. Por el contrario, si es mayor, se añadirá ochenta, como en la primera métrica, al vector de distancias.

Finalmente, se debe reducir el vector de distancias, al cual llamo d, a un único valor, al cual llamo err, para poder trabajar con él.

Esta distancia se considera que es el error de posición entre los puntos del patrón y los del láser. Este único valor lo puedo obtener a través de tres métodos distintos.

La incertidumbre sale de comparar la distancia entre las trazas del láser con los puntos del patrón, sumando a la coordenada X de los puntos del patrón el error de posición, ya sea positivo o negativo.

El método Q devuelve el error cuadrático medio, con la ecuación e.9.

$$err = \sqrt{\frac{sum(d^2)}{longitud\ de\ d}} \quad \text{(e.9)}$$

El método M devuelve el error medio.

El método H devuelve el error máximo.

En el apartado 4.1 del capítulo siguiente, se realiza un estudio para conocer la mejor métrica y el mejor modelo de cara a detectar las galerías.

3.4 Posicionamiento respecto a la galería

Por último, es necesario saber la posición del túnel en la que se ha realizado el patrón de puntos con el contorno de la galería. Esta posición no varía dependiendo de la manera en que se realiza el patrón, que puede ser de forma manual o de forma automática.

La posición desde la que se realiza el patrón es el punto de la intersección entre dos rectas imaginarias. Una de las rectas está centrada y paralela a las paredes del túnel, por el contrario, la otra recta esta también centrada y paralela a las paredes, en este caso de la galería. El punto verde, de la figura 11, que es la intersección entre las dos rectas rojas, es la posición desde la que se toma el patrón en esta supuesta galería.

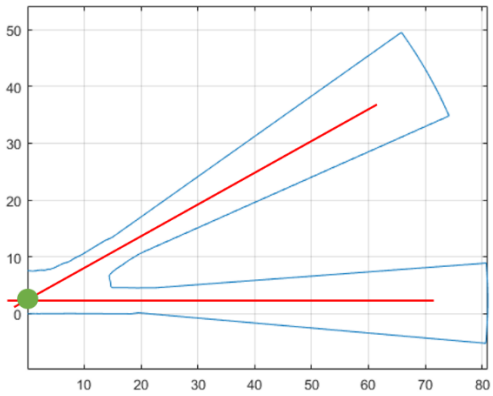

Figura 11: Posición respecto a la galería desde la que se realiza el patrón de puntos

Si se realiza el patrón de forma automática, la posición respecto a la galería desde la que se realiza el patrón de puntos es la misma, ya que en ese punto se toman los puntos de la traza. A estos puntos de la traza se les realizan diferentes filtrados para obtener el patrón de puntos.

Finalmente, el método proporciona la posición relativa. La posición relativa es la distancia que hay entre el robot y la posición respecto a la galería desde la que se realiza el patrón, con las coordenadas del robot como referencia. La posición relativa tiene un alcance de cinco metros antes de llegar al punto y cinco metros después. Añadir también, que esta posición relativa es buena cuando la incertidumbre, explicada anteriormente, es menor a 0.5m.

Se define como mapa del túnel al conjunto de galerías de las que se posee un patrón para identificarlas y una posición de referencia de donde se encuentran. Si se emparejan los puntos de la traza con el patrón y se conoce el mapa, se pueda localizar el robot.

Capítulo 4

Análisis de los factores que inciden en el método

4.1 Métrica

Se realiza un estudio para conocer qué métrica, para emparejar la traza con los puntos clave, es la que funciona mejor para detectar las galerías.

Los resultados que se obtienen, con cada métrica, con un patrón de la galería 15 se encuentran en la siguiente tabla, donde I es incertidumbre y P.R es la posición relativa.

Galería 15		
Métrica Polar	I: Próxima a 0 durante 70 muestras	P.R: Conocida de 4.9m a -5m
Métrica Nearest	I: Menor a 0,5 durante 120 muestras	P.R: Conocida de 5m a -5m
Métrica X	I: Próxima a 0 durante 13 muestras	P.R: Conocida de 1m a -3m

Tabla 1: Resultados de las métricas en un patrón de la galería 15

En la tabla 1, se puede ver como la métrica X es la que peor va, debido a que la posición relativa del robot se conoce durante una distancia menor y con apenas 13 muestras, que comparados con los 120 de la métrica Nearest es un resultado pobre.

Respecto a las otras dos métricas tampoco se puede diferenciar cuan es mejor, a causa de que son resultados bastantes parecidos. La única diferencia que se puede considerar es que la incertidumbre es menor a 0.5m durante más muestras en la métrica Nearest. Se realiza lo mismo, pero con un patrón de la galería 16, para sacar más conclusiones en claro.

Galería 16		
Métrica Polar	I: Próxima a 0 durante 22 muestras	P.R: Conocida de 2.7m a -0.2m
Métrica Nearest	I: Menor a 0,5 durante 50 muestras	P.R: Conocida de 3.4m a -3.2m
Métrica X	I: Se queda en un valor mínimo de 1.1	P.R: No es conocida

Tabla 2: Resultados de las métricas en un patrón de la galería 16

Nuevamente, la métrica X es la peor. En este caso, como se puede ver en la tabla 2, en ningún momento se conoce la posición relativa, porque la resta de las coordenadas Y de los puntos de la traza con los puntos del láser no es menor a 0.5m con todos los puntos del patrón. Para este patrón de la galería 15, la métrica Nearest va mucho mejor que la métrica Polar, ya que se conoce la posición relativa el doble de metros. La métrica Nearest se está postulando como la mejor para detectar las galerías.

Por último, se utilizar un patrón de la galería 17, para confirmar todo lo mencionado anteriormente.

Galería 17		
Métrica Polar	I: Próxima a 0 durante 115 muestras	P.R: Conocida de 1m a -4.7m
Métrica Nearest	I: Menor a 0,5 durante 209 muestras	P.R: Conocida de 4.6m a -4.5m
Métrica X	I: Próxima a 0 durante 96 muestras	P.R: Conocida de 0.5m a -4.53

Tabla 3: Resultados de las métricas en un patrón de la galería 17

Visualizando la tabla 3, se verifica que la mejor métrica es la Nearest, porque se conoce la posición relativa del robot durante una mayor distancia y, además, durante más muestras del método. La métrica X, sigue siendo la peor, y con la métrica Polar se puede llegar a detectar la galería, pero de una forma peor que la Nearest.

En la figura 12 se encuentran los datos de la tabla 3.

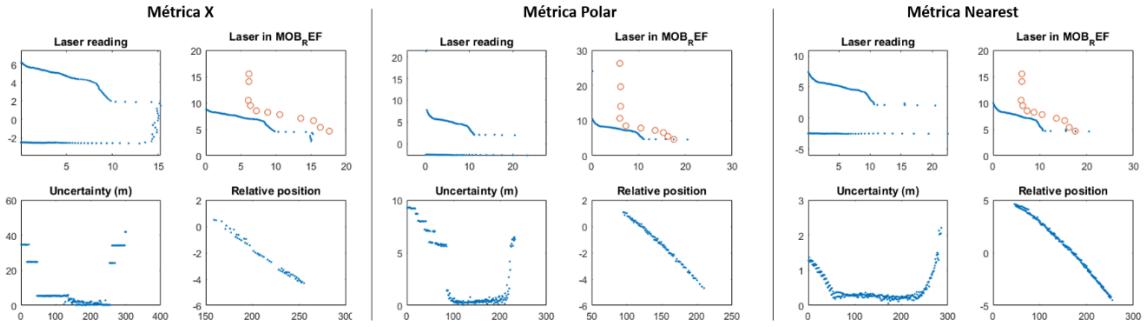

Figura 12: Resultados al utilizar cada una de las métricas

Una vez que se conoce cuál es la mejor métrica, se identifica cual es el mejor método para sacar la distancia a la que se encuentran los puntos del patrón de los puntos de la traza del láser. Para ello se usa la métrica Nearest con cada método de cálculo del error, y con el patrón de la galería 15. Donde mejor se ve el resultado es en las gráficas de relative position en la figura 13.

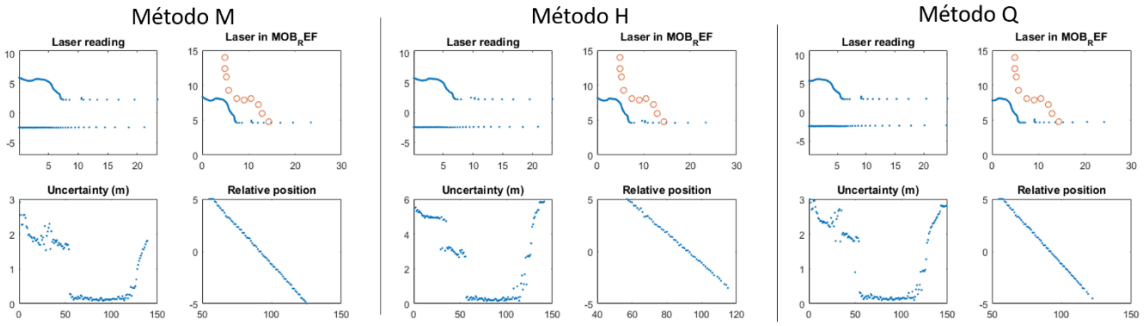

Figura 13: Resultados al utilizar cada una de los métodos para tener un único valor de distancia

Se puede ver como el peor método es el H, porque al quedarte con el valor máximo del vector de distancias antes se pierde la posición relativa. Entre el método M y el método Q apenas hay diferencia, pero si tienes que elegir uno, se ve mejor el método M, que te proporciona el error medio, porque la incertidumbre es próxima a cero durante unas pocas muestras más del método de detección de galerías y, por lo tanto, se conoce la posición relativa del robot hasta una distancia de cinco metros después de sobrepasar el punto de referencia de la galería, algo que con el método Q se conoce hasta una distancia de cuatro metros y medio.

4.2 Número y localización de los puntos del patrón

Para reconocer el número y localización de los puntos del patrón se realiza un estudio sobre ello tomando los puntos de forma manual con la métrica Nearest y el método M, los cuales son los mejores para detectar las galerías como se indica en el apartado anterior.

El estudio se ha considerado oportuno realizarlo con tres de las galerías del túnel de Somport, en concreto las galerías 17, 16 y 15.

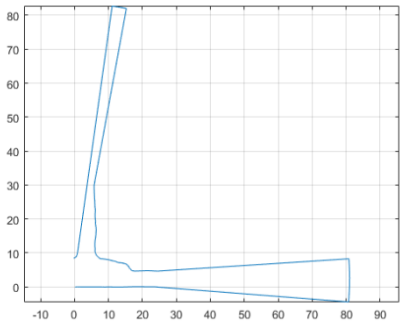

Figura 14: Traza galería 17

Respecto a la galería 17, cuya forma se ve en la figura 14, se considera apropiado empezar el estudio colocando diez puntos clave del patrón distribuidos aleatoriamente por el contorno de esta galería. Comparando estos puntos clave con los puntos obtenidos por el láser durante el tiempo en el que el robot está pasando por la galería, se obtiene una incertidumbre nula durante 120 muestras del método, es decir, durante esos momentos el robot sabe localizarse en todo momento.

Después se considera oportuno poner puntos clave en la parte izquierda de la galería, debido a que así se pueden detectar más zonas de la galería y localizarse durante más tiempo. Esta conclusión es errónea porque la incertidumbre es casi nula durante una cantidad de muestras menor a la anterior prueba. Esto se debe a que en los primeros y últimos momentos el láser no detecta la zona izquierda de la galería, siendo esta solo detectada cuando está el robot junto a ella. Tener esos puntos clave en la zona izquierda cuando esta no es detectada, produce un error de posición mucho mayor y, por lo tanto, una incertidumbre mayor. Debido a lo anterior, también es conocida la posición relativa durante una distancia menor.

Como conclusión a esto, los puntos clave se colocan en zonas que siempre que el robot se encuentre junto a la galería, estas sean detectados en todo momento y, así evitar lo anterior.

Posteriormente, se aumenta el número de puntos clave a trece sin poner ninguno en la parte izquierda, no se consigue ninguna mejora, ya que la incertidumbre y la posición relativa son prácticamente iguales a las obtenidas en el caso de diez puntos clave.

En cambio, si estos puntos pasan de estar separados por una distancia entre ellos a encontrarse mucho más próximos a la esquina derecha de la galería se consigue una mejoría, porque se detecta la posición relativa del robot mucho antes y la incertidumbre es casi nula durante más muestras del método. Con estos resultados se llega a la conclusión de que contra más cerca de la esquina están los puntos clave, durante más tiempo se localiza el robot, esto es debido a que el láser detecta la esquina todo el rato prácticamente igual, algo que no ocurre con las paredes

laterales de la galería que van cambiando dependiendo donde se encuentra el robot, ya que puede haber partes que hasta un momento dado no sean detectadas por el láser.

Si se colocan los puntos sin que ninguno de estos se encuentre en la esquina derecha de la galería salen unos resultados peores a los anteriores, ya que la posición relativa se conoce durante una menor duración, lo que reafirma la conclusión anterior de poner los puntos que conforman el patrón en zonas visibles por el láser del robot en todo momento.

Como estudio final de esta galería, se decide poner puntos únicamente en la esquina de la galería, lo contrario al caso anterior. Los resultados salieron mejores que todos los casos anteriores y, por lo tanto, se ve como las conclusiones anteriores se afirman. Aunque estos puntos tienen mejores resultados tienen un defecto, y es que al ser solo de la esquina pueden servir para detectar otras galerías, algo que no se desea.

Por lo tanto, la conclusión final es que hay que constituir el patrón con puntos que se encuentren en la parte derecha de la galería y cercanos a la esquina, pero sin estar todos en ella. La incertidumbre y la posición relativa del mejor de los casos se pueden visualizar en la figura 15, donde los círculos rojos son los puntos del patrón.

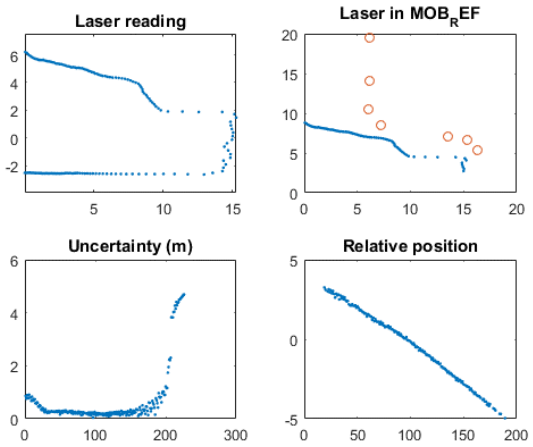

Figura 15: Resultados de la incertidumbre y la posición relativa con el patrón de puntos adecuados de la galería 17

Para finalizar con la galería 17 se incluye la tabla 4, donde se recogen todas las incertidumbres, I, y posiciones relativas, P.R, obtenidas en todos los casos planteados anteriormente.

Galería 17	
Con diez puntos clave distribuidos uniformemente	
I: Próxima a 0 durante 120 muestras	P.R: Se conoce de 1m a -5m
Con diez puntos clave distribuidos uniformemente y dos puntos a la izquierda	
I: Próxima a 0 durante 50 muestras	P.R: Se conoce de 1m a -1.5m
Con trece puntos clave distribuidos uniformemente	
I: Menor a 0.5 durante 120 muestras	P.R: Se conoce de 1m a -5m
Con los 7 puntos clave próximos a la esquina de la galería	
I: Próxima a 0 durante 170 muestras	P.R: Se conoce de 3.5m a -5m
Sin puntos en la esquina de la galería	
I: Menor a 0.5 durante 110 muestras	P.R: Se conoce de 0.8m a -5m
Con 6 puntos clave justo en la esquina de la galería	
I: Menor a 0.5 durante 190 muestras	P.R: Se conoce de 4m a -5m

Tabla 4: Diferentes casos con la galería 17

Respecto a la detección de la galería 16 lo primero que se hace es comprobar que los puntos finalmente seleccionados de la galería 17 no detecten la nueva galería, ya que sino la opción escogida en la galería 17 no es la idónea porque no detecta una única galería.

La forma de la galería 16 se encuentra en la figura 16.

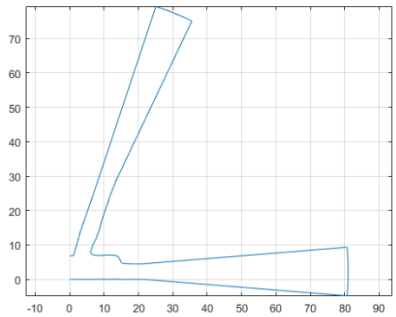

Figura 16: Traza galería 16

Esta comprobación es exitosa porque los puntos clave de la galería 17 en la galería 16 no llegan a una incertidumbre próxima a cero, llegando como máximo a una incertidumbre de uno. Debido a esto, tampoco se tiene una posición relativa del robot, ya que está limitada a darla cuando la incertidumbre es menor a un 0.5m. Con lo mencionado anteriormente, se puede decir que los puntos clave de la galería 17 están bien tomados.

Después, se toman diez puntos de la parte derecha de la galería distribuidos entre sí, ocupando toda la galería incluso la pared derecha de esta. Con estos puntos durante unas pocas muestras se encuentra con incertidumbre nula o casi nula. La incertidumbre no es nula durante más muestras porque hay puntos clave que se encuentran en la parte interna de la galería. Estos puntos, debido al ángulo en el que se encuentra la entrada de la galería, solo son detectados justo cuando el robot está posicionado delante de ella. Por lo tanto, se considera oportuno quitar los puntos internos que se encuentran muy alejados de la entrada de la galería.

Al quitar estos puntos, se aprecia como la detección es mucho más eficiente. Esto se debe a que ahora se detectan todo el rato los puntos que conforman el patrón, aunque sea con un error de posición. La incertidumbre se queda fija en un valor casi nulo durante una mayor duración que en el caso anterior.

Si se eliminan otros dos puntos de la zona interna de la galería y se añade uno en la esquina y otro en la pequeña abertura de la galería, se obtiene prácticamente el mismo resultado, pero con este patrón de puntos la galería se detecta con incertidumbre nula un poco antes. Por lo tanto, se puede conocer la posición relativa del robot mucho antes.

Para afirmar más aún que es erróneo poner puntos clave en la parte izquierda de la galería, se coloca un único punto a la izquierda de esta, junto al patrón del caso anterior, para ver que ocurre. Aunque, solo se introduce un único punto a la izquierda, el resultado es peor que en el caso anterior porque la incertidumbre es nula durante un menor número de muestras del método. Por lo tanto, sigue perjudicando el hecho de colocar puntos clave en la zona izquierda de la galería.

Como última prueba del estudio de la galería 16 se quitan los puntos de la pequeña abertura que posee dicha galería, a ver si la detección mejora o empeora, dejando solo los de la esquina de la galería. Comparando los resultados obtenidos con el mejor patrón de esta galería, que es

el obtenido hace dos casos, apenas se ven mejoras, incluso puede que de esta forma sea incluso peor.

El número de puntos clave que conforman el patrón, mientras estén en zonas visibles en todo momento, pueden variar entre ocho y quince puntos clave e incluso más. Esto se puede afirmar porque incrementando el tamaño del mejor patrón anteriormente explicado a un número de quince puntos, salen prácticamente los mismos resultados.

La conclusión final respecto a esta galería es que hay que tener los puntos más próximos entre sí ,y en una zona que el robot la vaya a detectar todo el rato como son las aberturas que hay en la entrada de la galería y la esquina, ya que si se colocan en las paredes internas de la galería estos no se ven por la angulación que hay en la entrada de esta. Dentro de esta zona se pueden poner tantos puntos clave, que conforman el patrón, como quieras.

Los resultados del mejor patrón obtenido y que reflejan la conclusión final se encuentran en la figura 17.

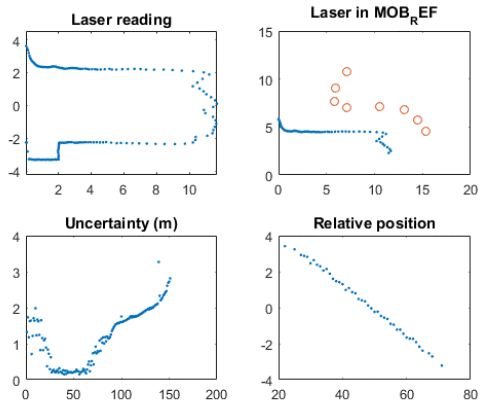

Figura 17. Resultados de la incertidumbre y la posición relativa con el patrón de puntos adecuados de la galería 16

La tabla 5 contiene todos los casos propuestos.

Galería 16	
Con los puntos clave de la galería 17	
I: Próxima a 0 durante 0 muestras	P.R: No se conoce
Con los 10 puntos clave distribuidos uniformemente entre sí	
I: Menor a 0.5 durante 8 muestras	P.R: Se conoce de 2m a -2m
Con los 8 puntos clave un poco más próximos a la esquina y la abertura	
I: Próxima a 0 durante 50 muestras	P.R: Se conoce de 3m a -3m
Con los 8 puntos clave distribuidos próximamente entre ellos en la esquina y la abertura	
I: Menor a 0.5 durante 50 muestras	P.R: Se conoce de 3.8m a -3m
Con los mismos puntos que la gráfica 12 más un punto clave en la zona izquierda	
I: Próxima a 0 durante 25 muestras	P.R: Se conoce de 3.8m a -1.7m
Con los 4 puntos clave distribuidos en la esquina de la galería	
I: Menor a 0.5 durante 50 muestras	P.R: Se conoce de 3.2m a -2.5m
Con los 15 puntos clave distribuidos próximos entre sí en la esquina de la galería y la abertura	
I: Próxima a 0 durante 60 muestras	P.R: Se conoce de 3.8m a -3.5m

Tabla 5: Diferentes casos con la galería 16 donde I y P.R son las abreviaturas de incertidumbre y posición relativa

La última galería que se utiliza en el estudio es la galería 16, la cual aparece en la figura 18.

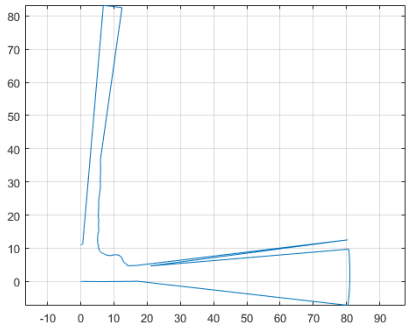

Figura 18: Traza galería 15

Respecto a la galería 15 se considera, como en los casos anteriores, distribuir diez puntos entorno a la galería, aunque se sabe que por las conclusiones de las dos anteriores galerías se acabarán juntando más los puntos. Se realiza otro caso con estos puntos mucho más próximos entre sí.

Como era de esperar sale mucho mejor el segundo caso ya que, se consigue una mejor detección contra más cerca están los puntos de la esquina de la galería porque es la zona más visible. El resultado de poner los puntos más próximos entre sí sale bastante bueno en vista de que se está detectando la galería en todo momento y se conoce la posición relativa del robot cinco metros antes y después de pasar por el punto de referencia de la galería, que es desde el cual se toma el patrón de puntos.

Las galerías 16 y 15 son muy parecidas entres sí, por ello se decide probar los puntos clave obtenidos en la galería 16 para ver si estos no sirven en la galería 15, que es lo que debe pasar y es justo lo que pasa.

Tampoco se consigue tener una posición relativa entorno a la galería 15 con el patrón final de puntos de la galería 17.

Bajando el número de puntos en el patrón de diez a seis, sigue saliendo un buen resultado, pero un poco peor que el obtenido con diez, como se puede apreciar en los resultados encontrados en la Tabla 6. Los mejores resultados se encuentran en la figura 19.

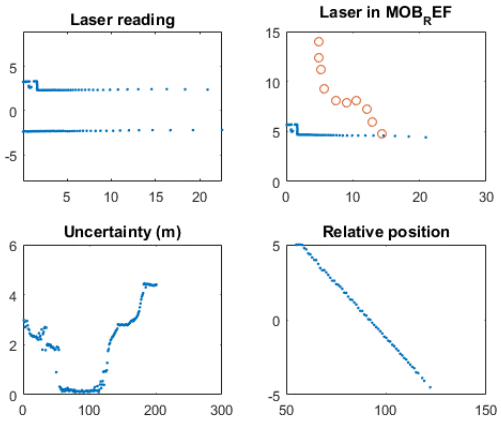

Figura 19: Resultados de la incertidumbre y la posición relativa con el patrón de puntos adecuados de la galería 15

Por lo tanto, es mejor para detectar cuando se crea un patrón conformado por entorno a diez puntos clave. Todos los casos planteados en el estudio de la galería 15 se en encuentran en la tabla 6.

Galería 15	
Con 10 puntos clave alejados	
I: Menor a 0.5 durante 25 muestras	P.R: Se conoce de 2m a -1.6m
Con 10 puntos clave próximos	
I: Próxima a 0 durante 70 muestras	P.R: Se conoce de 5m a -5m
Con los 8 punto clave de la galería 16	
I: Menor a 0.5 durante 0 muestras	P.R: No se conoce
Con los 8 punto clave de la galería 17	
I: Próxima a 0 durante 0 muestras	P.R: No se conoce
Con 6 puntos claves	
I: Menor a 0.5 durante 65 muestras	P.R: Se conoce de 5m a -4m

Tabla 6: Diferentes casos con la galería 15 donde I y P.R son las abreviaturas de incertidumbre y posición relativa

Con estos tres estudios se puede concluir que la cantidad de puntos que constituyen el patrón debe estar entorno a diez, pero estos deben encontrarse en la parte derecha de la galería, y en una sección donde se puedan detectar en todo momento mientras el robot pasa por dicha galería.

4.3 ¿Vale para apartaderos y refugios?

Los túneles poseen, aparte de las galerías, otros elementos estructurales para mejorar la seguridad en ellos como son los apartaderos y los refugios. Se va a comprobar si el método de detección de galerías sirve también para detectar alguno de estos dos elementos estructurales.

Primero se tiene en cuenta que los todos los apartaderos de un túnel tienen más o menos la misma forma, algo que no ocurre con las galerías. Los refugios de un mismo túnel también tienen las mismas características. Por lo tanto, se supone que con un mismo patrón de puntos podremos detectar cada uno de ellos, pero no se pueden diferenciar por los puntos que nos proporcione el láser. Si se posee del mapa del túnel, sí que se puede diferenciar los apartaderos y los refugios, porque se puede mirar la posición y cuantos hay entre una galería y otra y, por lo tanto, se pueden numerar.

Se realiza un patrón de puntos de un apartadero, teniendo en cuenta las conclusiones sacadas de los estudios del apartado anterior. El patrón de puntos se encuentra en la figura 20.

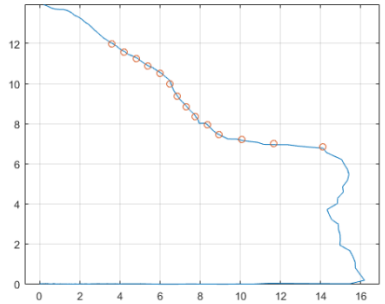

Figura 20: Patrón de puntos en un apartadero

Como con las galerías, se utiliza la métrica Nearest y el método M para el error de posición, salieron unos resultados favorables, ya que se conoce la posición relativa cinco metros antes del punto central del apartadero, y cinco metros después. La incertidumbre solo es próxima a cero mientras el robot se encuentra al lado del apartadero. Por lo tanto, se puede concluir que este método también puede servir para identificar los apartaderos.

Respecto a los refugios, se realiza lo mismo que para el caso anterior. Se toma un patrón bueno y se utiliza la métrica Nearest y el método M. Los resultados como se pueden ver en la figura 21 son decepcionantes.

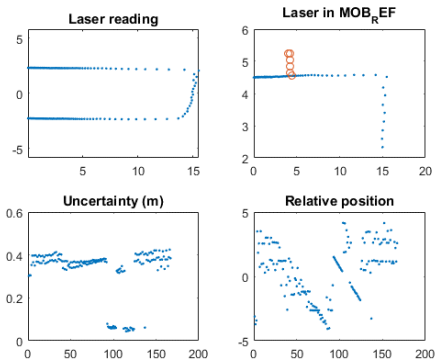

Figura 21. Patrón de un refugio y utilizando la métrica Nearest

Los resultados son malos porque el ancho del túnel sin refugio es un poco menor a cuando sí que lo hay y, por lo tanto, debido a la cercanía el método nos devuelve una incertidumbre menor a 0.5 durante toda la simulación. Utilizar el método Nearest para la detección de refugios queda descartado.

Con la métrica Polar ocurre exactamente lo mismo, así que también queda descartada.

Finalmente, con la métrica X y utilizando el mismo patrón, el refugio solo se detectaba una única vez, justo en la posición desde la que se había tomado el patrón, pero ya es un resultado mejor que los anteriores. Solo se detecta una única vez porque cuando uno de los puntos que conforma el patrón se encuentra justo encima de los puntos del láser que simulan el muro del túnel se obtiene una distancia grande, debido a que esta métrica devuelve la distancia del punto del patrón a uno que se encuentre a su misma altura, caso que ocurre con el punto de la traza que se encuentra en la posición cero de la coordenada X. Para solucionar esto, se decide que el patrón de puntos no debe contener ningún punto a la misma altura que los puntos del láser que simulan la pared. También, la diferencia entre los puntos de la traza y del patrón en la coordenada Y, se ha reducido de 1.5m a 0.5m. Con estos cambios, los resultados son buenos. Los resultados se encuentran en la figura 22.

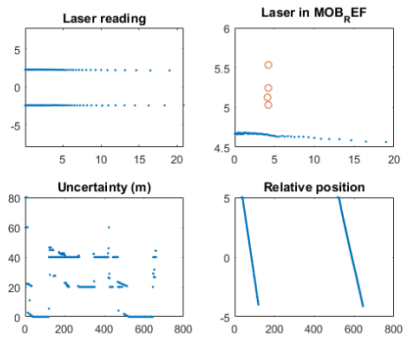

Figura 22: Patrón de un refugio y utilizando la métrica X

Comparando la figura 22 con la21, se puede ver como el patrón, que son los puntos rojos, ya no se encuentra ningún punto a la misma altura en la coordenada Y que los puntos de la traza que simulan la pared del túnel. Se puede ver como la posición relativa en la figura 22 se conoce durante cinco metros antes y cuatro metros después del punto de referencia del refugio, su posición intermedia, algo que en la figura 21 con la métrica Nearest no se conoce. En esta misma gráfica de la figura 22 se puede ver como se conoce la posición relativa durante dos momentos distintos, ya que se ha dejado la simulación hasta que el robot ha pasado por dos refugios distintos.

Concluir, que solo se puede utilizar únicamente este patrón y esta métrica a la hora de detectar los refugios, y si se cuenta con un mapa del túnel para poder localizarse.

4.4 Automatización de la obtención de los puntos del patrón

Se realiza un método para obtener el patrón automáticamente y perder mucho menos tiempo que de la forma manual. También se puede aplicar en el robot y que este mismo cree los diferentes patrones. En el método se tienen en cuenta las conclusiones sacadas de los estudios del apartado 4.2.

Para realizar la obtención de los puntos clave que conforman el patrón, primero se deben saber las diferencias que hay cuando un robot se encuentra en una zona del túnel donde no hay ninguna galería, a cuando es una zona que sí que tiene. Los vectores de puntos que se obtiene en ambos casos para compararlos son los siguientes:

a_tunelSinGaleria=[1.096736946534270 0.804432818284997 0.599778377235050 0.420956808801550 0.482814590584315 0.356120967939646 0.288229484945621 0.255384998737106 0.221570792843225]

r_tunelSinGaleria=[5.154416067800503 6.315607650891559 8.149388259249893 11.296174396670760 9.940253568194326 13.251445204203200 16.323360560864913 18.272653228253418 20.931708004842800]

a_tunelConGaleria=[1,04924178630478 1,02520515034336 0,993639069156216 0,242660786483142 0,969994422229284 0,938554431758007 0,878920061951470 0,785256097764081 0,691613396918425 0,605606260188335 0,518697281242551 0,437968668420747 0,369216839271237]

r_tunelConGaleria=[17,8191297206121 16,2490222475077 13,9376621066806 7,32459555197418 11,6149867412753 10,6048721821623 9,75260908680339 9,95464936599979 10,8579645422151 12,3212457162415 14,0938202415101 15,5364030908058 15,7085983142991]

Como era de esperar, se puede apreciar como para un ángulo de casi $\pi/2$, la distancia, que hay hasta la pared en la zona donde hay galería, de algunos puntos es mayor a cuando no la hay. Por lo tanto, lo primero que hay que hacer para obtener los puntos del patrón es saber si el robot se encuentra junto a una galería o no. Esto se consigue comparando todos los puntos de la traza obtenidos en todo momento. Si alguno de estos puntos tiene una distancia mayor o igual a 12m y un ángulo mayor a $\pi/4$, significa que nos encontramos al lado de una galería y, por lo tanto, se realiza el método de creación del patrón. En caso de que no exista ningún punto con esas características, el robot se encuentra entre las paredes del túnel y no se realiza nada.

Los puntos clave se toman desde el punto de referencia de cada galería. Si este no se conoce, para cumplir la conclusión de que no formen parte del patrón puntos que se encuentran en la parte izquierda de la galería, y que se tomen los puntos del patrón más cerca del lateral derecho de esta, se introduce una condición, que delimita la obtención de puntos clave para cuando todas las trazas del láser se encuentran en una posición en la coordenada X respecto al robot

mayor a 0.5m, y en una posición en la coordenada Y respecto al robot mayor a 10m, un valor mucho mayor a la distancia que suele haber entre las dos paredes del túnel.

Después de saber la posición desde la que se toma el patrón, se guardan los puntos de la traza una única vez por galería, gracias a una variable, en un vector todas las posiciones en la coordenada X y en otro las posiciones en la coordenada Y. Se tiene que realizar una filtración de dichos puntos guardados, para obtener al final una pequeña cantidad, pero a la vez buena para detectar la galería. Como ya se explica anteriormente, el patrón debe estar conformado por unos quince puntos.

El primer filtrado borra todos los puntos cuya distancia en la coordenada X, referida al robot, sea menor a 3.5m, lo cual elimina todos los puntos que haya cogido de la parte izquierda de la galería.

El segundo filtrado elimina todos los puntos que tienen una distancia en la coordenada Y mayor a 12m, para así eliminar puntos lejanos que no son detectados por el láser en todo momento. Los puntos restantes se convierten a coordenadas polares para realizar dos filtrados más.

El siguiente filtrado elimina todos los puntos que tengan una coordenada radial mayor a 20m, y junto a otro filtrado que elimina puntos cuya coordenada angular es menor a $\pi/8$, se eliminan puntos lejanos al robot o que se encuentran prácticamente justo delante de él. De esta manera, la gran mayoría de los puntos restantes se encuentran en la zona derecha de la entrada de la galería espaciados entre ellos.

Aunque, los puntos se encuentran acumulados en una misma zona, sigue habiendo muchos. Para reducir el número, se realiza un filtrado final en el que se borran tres de cada cuatro puntos, si había menos de sesenta puntos en total. En caso contrario, se borran cinco de cada seis puntos. Al final, quedan entorno a diez o catorce puntos, que son los que conforman el patrón.

Los patrones de puntos quedan entorno a la parte derecha de la esquina de la galería, ya que en el estudio anterior se ve que los puntos deben estar en zonas visibles en todo momento.

En las figuras 23 y 24 se pueden ver los puntos que conforman la galería junto con el patrón de puntos sacado automáticamente.

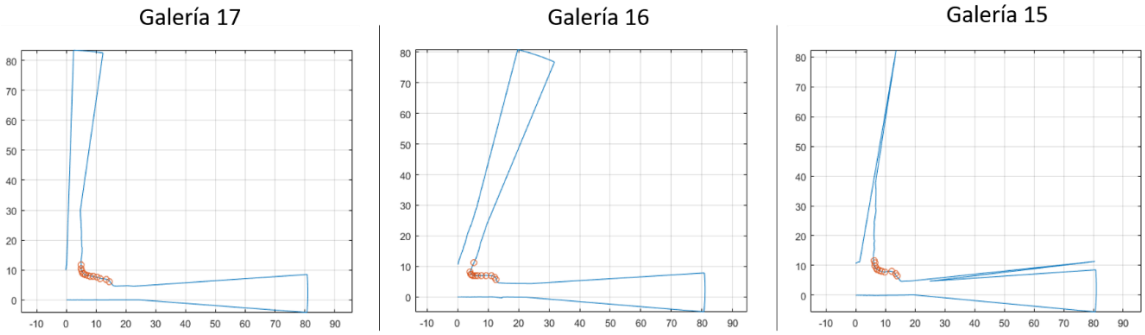

Figura 23: Galerías junto a patrones de puntos

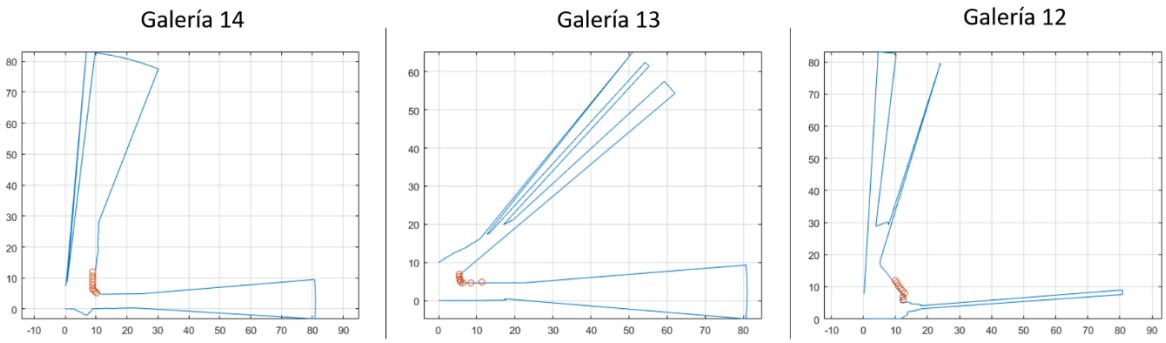

Figura 24: Otras galerías junto a patrones de punto

Finalmente, al aplicar el método de detección de galerías junto con los patrones obtenidos automáticamente, salen los resultados que se encuentran en la tabla 7, los cuales son buenos.

Galería	Incertidumbre	Posición relativa
Galería 17	Próxima a 0 durante 200 muestras	Se conoce de 4m a -5m
Galería 16	Menor a 0.5 durante 75 muestras	Se conoce de 4m a -2m
Galería 15	Próxima a 0 durante 75 muestras	Se conoce de 5m a -5m
Galería 14	Menor a 0.5 durante 40 muestras	Se conoce de 5m a -2m
Galería 13	Próxima a 0 durante 70 muestras	Se conoce de 5m a 5m
Galería 12	Menor a 0.5 durante 90 muestras	Se conoce de 5m a -5m

Tabla 7: Incertidumbres y posiciones relativas de cada comparación

Para poder realizar de forma automática los patrones de puntos, el láser debe tener un rango mínimo de 15m.

Capítulo 5

Implementación de Matlab a ROS

5.1 ¿Qué es y en que consiste ROS?

ROS (*Robot Operating System*, Sistema Operativo Robotico) **[5]**, es un *framework* flexible para realizar aplicaciones para robots. Es una colección de librerías o herramientas para simplificar el procedimiento de crear un comportamiento robusto y complejo de los robots en una amplia variedad de plataformas robóticas, porque crear un software realmente robusto es difícil.

Este sistema se creó porque los problemas que parecen triviales para los humanos, para los robots no son tan sencillos, debido a las tareas o ambientes. Así, se pone toda la información en común y se llega a metas que un individuo por su cuenta le costaría alcanzar. Como resultado, ROS se creó para incitar el desarrollo colaborativo de software de robótica.

Aunque no es un sistema operativo, ROS provee los servicios de uno de estos como herramientas de visualización, comunicación por mensajes, abstracción de hardware y administración de paquetes entre otras cosas.

Está basado en una arquitectura de grafos compuesto por nodos, que pueden mandar, recibir y multiplexar mensajes de sensores, como de láseres, de estados o actuadores entre otros.

Lo único malo es que esta librería está orientada para un sistema UNIX como es Ubuntu y, por lo tanto, no se puede usar en sistemas operativos como Windows. Si lo quieres usar en este tendrás que realizar una partición de disco duro para poder instalarte en ella Ubuntu, o crear una máquina virtual, una opción mala, ya que entonces se trabaja mucho más lento. Ubuntu es un sistema distribuido bajo una licencia libre, con un fuerte enfoque en mejorar la experiencia del usuario y es una distribución Linux.

ROS también contiene una suite, llamada ros-pkg, donde se encuentran paquetes aportados por otros usuarios. En esta sección, se pueden encontrar paquetes con diferentes funciones que sirvan para tu proyecto como localización, simulación o planificación. En ROS se puede utilizar los lenguajes de programación Python y C++.

Algunos de los conceptos más importantes de ROS, que quedan representados en la figura 25, son:

- Nodos: Los nodos son programas ejecutables que realizan una determinada función, por ejemplo, un cálculo. Los nodos se combinan en un grafo y se comunican entres sí a través de *Topics*. Cada nodo del grafo está destinado a realizar una función independiente. Un nodo puede controlar los motores de las ruedas y otro mientras realiza la localización. Esto se puede realizar, ya que se compilan individualmente y son gestionados por un nodo principal como es el *ROS Master*. Debido a la comunicación que hay entre los nodos mediante *Topics*, los proyectos son más fáciles de realizar en comparación con los programas de un solo nodo. Un nodo en ROS es escrito con el uso de librerías clientes Ros, como son roscpp, si se utilizar el lenguaje de programación C++, o rospy, si se utiliza Python.

- Ros Master: Proporciona servicios de nombre y registro al resto de nodos que se están ejecutando. Realiza un seguimiento de los *publishers* y *subscribers*. Su función es habilitar nodos entre sí, es decir, que se localicen unos a otros. El Master se ejecuta usando el comando roscore, el cual carga el ROS Master junto a otros componentes importantes.

- *Topics*: Los nodos se comunican gracias a estos, intercambiando mensajes. Poseen una comunicación unidireccional en tiempo real. La semántica de los *topics* se basa en publicar o suscribir (*publishers* y *subscribers*). Los nodos que publican no saben en realidad cuantos nodos están leyendo esta información, pero en cambio sí un nodo quiere recibir información, debe suscribirse al tema relevante. Un *topic* puede tener múltiples suscripciones o publicaciones. Los mensajes utilizados para la comunicación pueden incorporar tanto datos primitivos, como son enteros, *bool*, como *arrays* o estructuras.

- Servicios: Son transacciones síncronas entre nodos con una arquitectura cliente/servicio. Hay dos mensajes, uno es la solicitud y otro la respuesta. El nodo servidor solo realiza un procedimiento cuando el nodo cliente realiza una solicitud.

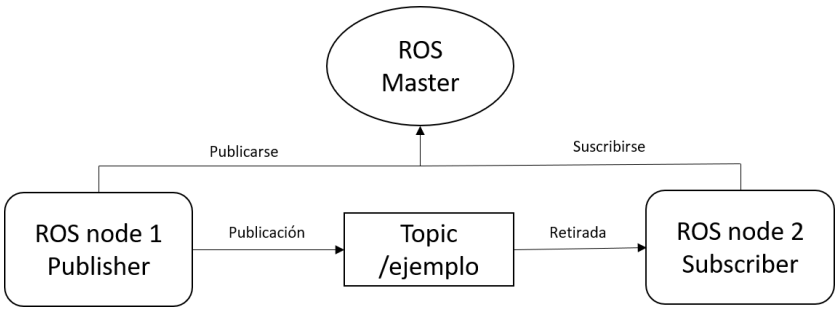

Figura 25: Ejemplo con algunos de los conceptos de ROS

5.2 Adaptación del método a ROS

En primer lugar, el lenguaje de programación que se utiliza es Python, en concreto Python 2, ya que muchas de las funciones de Python 3 no las reconoce ROS. Se utiliza la extensión de Google llamada Colaboratory, la cual te da la opción de programar en Python 2, para realizar la adaptación de las diferentes funciones y métodos.

Para empezar, se crea el entorno de trabajo o *workspace*, donde se guarda el paquete a crear. Se crean la carpeta *workspace* y dentro de esta la carpeta src, que se pueden crear por terminal con el comando c.1.

Mkdir -p ~/workspace/src (c.1)

Después, en el directorio /workspace, que como su propio nombre indica es el entorno de trabajo, se escriben los comandos c.2y c.3, para hacer *make* a este directorio.

Cd ~/workspace/ (c.2)
Catkin_make (c.3)

Al escribir estos comandos en la terminal se crea el fichero CMakeLists.txt, además de las carpetas build y devel, y el entorno de trabajo estará listo. Es de tipo catkin.

Dentro de la carpeta src se crea el paquete tfg que depende de la librería rospy, con los comandos c.4 y c.5.

Cd ~/workspace/src (c.4)
Catkin_create_pkg tfg rospy (c.5)

Posteriormente, se vuelve a aplicar el comando c.3 en el directorio de trabajo y se crear los archivos package.xml y CMakeLists.xml automáticamente.

En el interior de la carpeta tfg también se crean de forma manual las carpetas *launch*, donde se encuentra el lanzador de archivo en lenguaje de programación xml, y src, donde se encuentran los archivos necesarios para implementar el método de detección de galerías. Estos dos archivos se llaman Scan y Deftfg.

En la carpeta *launch* se crea el lanzador, que en este caso se llama tfg.launch. Este archivo se encarga de lanzar el nodo de un paquete y contiene las siguientes líneas:

```
<launch>
<node pkg="tfg" type="scan.py" name="tfg_test" output="screen">
</node>
</launch>
```

En estas líneas se indica el nombre del nodo, su tipo que debe corresponder a un ejecutable, en este caso el archivo principal scan.py, y el paquete donde se encuentra.

Los contenidos más importantes de cada archivo se explican a continuación.

5.2.1 Scan.py

Inicialmente se crea un archivo .py, el cual se nombra scan, en el que va la parte principal del método. Este script va encabezado por:

#! /usr/bin/env Python

Al poner esto se garantiza que el intérprete a utilizar sea el de Python.

Posteriormente, se necesita importar la librería rospy, para poder crear los nodos, *subscribers* o *publishers* que se vayan a utilizar en lenguaje de programación Python. Otras librerías que se necesitan importar son la numpy **[6]**, como np, para poder trabajar con matrices y vectores y la librería math **[7]**, para poder realizar diferentes operaciones matemáticas como cosenos, por ejemplo.

También, se importa *LaserScan* de la librería sensor_msgs.msg, para poder leer la información que proporciona el láser frontal del robot, que es el que se necesita. Gracias a esta información, se conoce el array de rangos en tiempo real que proporciona el láser, así como, el ángulo máximo, el ángulo mínimo o su incremento. La estructura compactada de la información que proporcionan los mensajes *LaserScan*, es la siguiente:

Std_msgs/Header header
Float32 angle_min
Float32 angle_max
Float32 angle_increment
Float32 time_increment
Float32 scan_time
Float 32 range_min
Float32 range_max
Float32[] ranges
Float32[] intensities

Por último, se importa de la librería std_msgs.msg el archivo *String*, para así poder realizar *publishers* de mensajes de este tipo de dato.

Una vez importado todo esto, se inicializa el nodo, que en el caso presente se llama test_tfg, con el comando c.6.

Rospy.init_node('test_tfg') (c.6)

La suscripción a un *topic* se realiza con el comando c.7.

Sub=rospy.Subscriber('/laser_front/scan', LaserScan, callback) (c.7)

En el primer argumento de esta función se escribe el *topic* al que te quieres suscribir, en este caso al /laser_front/scan para poder conocer en tiempo real la información que proporciona el láser frontal y poder detectar las galerías. El segundo argumento indica de que tipo es el mensaje, en este caso sensor_msgs.msg.LaserScan. El tercer argumento es la función que se invoca, en este caso llamada callback, cuando se recibe un nuevo mensaje. Este mensaje es el primer argumento de la función que se invoca.

Se escribe también el comando c.8, que evita que se salga del nodo hasta que el nodo ha sido cerrado y, a diferencia de roscpp, no afecta a la función que se invocaba cuando se produce una suscripción.

rospy.spin() (c.8)

En este script también se crea un objeto para publicar información a un *topic* gracias al comando c.9.

Pub=rospy.Publisher('informacionTunel, String) (c.9)

El primer parámetro es el topic en el que se publican los mensajes, en este caso de tipo std_msgs.msg.String, que lo indica el segundo parámetro. A través del objeto pub que se crea con c.9, se pueden publicar mensajes con el comando c.10. Los mensajes que se publican contienen información de cuál es la última galería detectada, cual es la siguiente a detectar y si justo se está detectando, como se ve en la figura 26.

pub.publish(str) (c.10)

Figura 26: Información que se publica

El script también contiene la definición de la función *callback* mencionada anteriormente. Como se comenta antes, esta tiene como primer argumento msg, que es la información que recibe gracias a la subscripción al *topic* /laser_front/scan.

Dentro de esta función, no se puede realizar ningún tipo de bucle, como la sentencia *for*. Por lo tanto, para poder adaptar todo el método propuesto en MATLAB, se tiene que realizar a parte otro archivo de Python donde estén definidas todas las funciones necesarias. Este archivo se explica más adelante. Algo de destacar de la función de llamada es que gracias a su primer parámetro podemos conocer todos los rangos de la traza. También, se consiguen todos los ángulos de la traza porque conocemos el ángulo mínimo, el ángulo máximo y el incremento del ángulo, y se puede realizar un array de todos los ángulos gracias a la función arange **[8]** de la librería numpy.

```
angles=np.arange(msg.angle_min,(msg.angle_max+msg.angle_increment),msg.angle_increme
nt)
```

Se realiza hasta un incremento mayor que el ángulo máximo porque este último es el límite y no se tiene en cuenta a la hora de crear el array. Si no se pone este incremento, el array de ángulos tiene una longitud de una unidad menor al array de rangos y, por lo tanto, se produce un error que se solventa de esta manera.

Para poder implementar el comando de MATLAB fminbnd, que encuentra el mínimo de una función cambiando automáticamente el primer parámetro de esta en un intervalo fijo, [5,-5] en este caso porque es el máximo a conocer de la posición relativa, se necesita importar optimize **[9]** de la librería sicipy. La función fminbound de optimize funciona de la misma manera que fminbnd. Referido a esto, el find de MATLAB se puede realizar en Python a través de in enumerate **[10]** como se indica a continuación:

MATLAB: I = find (errores_min < 0.5) ;

Python: I = [i for i,x in enumerate(errores_min) if x<0.5]

Hay que importar el archivo deftfg para poder utilizar las definiciones realizadas en él. Toda variable que se vaya a utilizar en estas funciones debe ser global **[11]**, ya que sino no se reconocen ni se puede variar su valor.

5.2.2 Deftfg.py

Este archivo contiene todas las definiciones de las funciones a utilizar para realizar el método de detección de galerías en el archivo scan.py. Aquí, también están importadas las librerías numpy y math.

Este archivo contiene las funciones sumaListayEscalar y guardarEnLista. Ambas contienen un *for*, como se puede ver en Anexos, para conseguir el resultado deseado. Estos *for* se realizan dentro de funciones porque no se pueden poner directamente en la función de llamada, *callback*, como se dice anteriormente.

Otra cosa a destacar es que en Python no existe la estructura de control *switch-case* **[12]**. La forma más factible de implementar esta estructura es mediante la asignación de diccionarios. Un diccionario **[13]** es una palabra que puede tener asociada un valor constante, como enteros o *arrays*, o referirse al nombre de una función entre otros. Se crean diccionarios de nombres de funciones como valores poniendo estos entre llaves. Los valores introducidos deben llamarse

igual que alguna de las funciones ya definidas. Debido a que en MATLAB se realizan dos *switch-case*, uno de métricas y otro de métodos para sacar el error de posición, se tienen que realizar dos diccionarios. Saber el valor de lo que devuelve cada función es sencillo, ya que existe la operación diccionario.get('key'), que devuelve el valor que corresponde con la *key* introducida, que en el caso presente es el nombre de una de las funciones.

La función de MATLAB dsearchn, no existe como tal para Python. Para poder adaptarla hay que importar cKDTree **[14]** de la librería sicipy.spatial. Se necesitan crear dos matrices, A y B. La matriz A posee dos columnas, en la primera los puntos de la traza del láser en la coordenada X y en la segunda en la coordenada Y. Lo mismo ocurre con la matriz B, pero esta posee los puntos que conforman el patrón. Se crea un objeto de la clase cKDTree que contiene a la matriz A. Finalmente, se aplica la función query en este objeto de la clase cKDTree. La función nos proporciona la distancia mínima, como realiza dsearchn, con un punto de la matriz, en este caso B, que se ponga como parámetro en la función query.

El grafo de la figura 27 muestra la implementación del método en ROS.

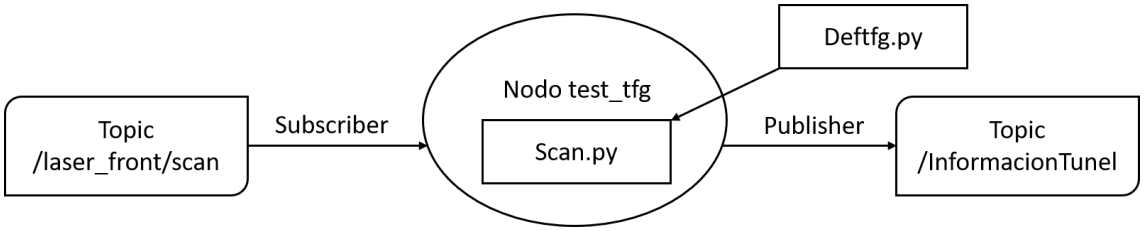

Figura 27: Implementación del método en ROS

Capítulo 6

Pruebas

6.1 Trazas reales Somport en MATLAB

Las primeras pruebas realizadas fueron mediante dos archivos .mat, ranges2.mat y angles2.mat, que contienen los rangos y los ángulos de trazas reales de Somport.

Todos los estudios realizados anteriormente fueron realizados utilizando estos dos archivos.

Para comprobar el perfecto funcionamiento del método, se realiza un fichero de MATLAB en la que se van cambiado los patrones de puntos clave automáticamente cuando la galería es detectada. Es aplicada tanto con patrones tomados de forma manual, como con patrones tomados de forma automática.

6.1.1 Patrones de puntos tomados de forma manual

Los resultados concluyentes se encuentran en la figura 28.

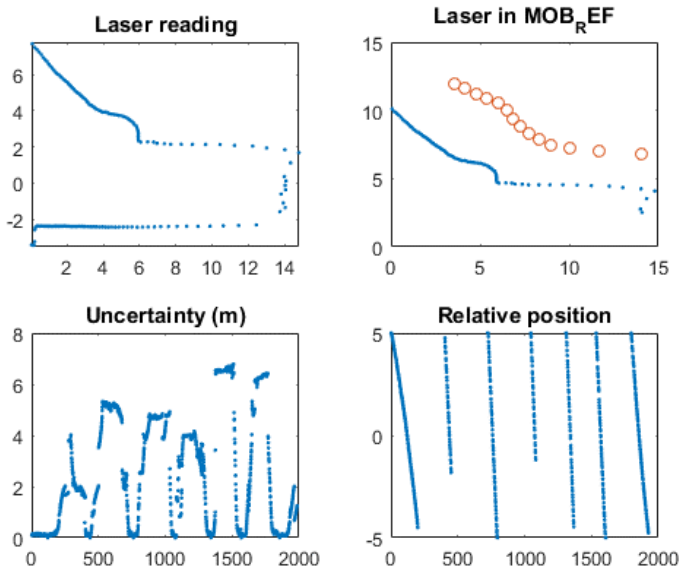

Figura 28: Resultados utilizando patrones tomados de forma manual

Como se puede apreciar todas las galerías, de la 17 a la 11, son detectadas perfectamente, algunas con más precisión que otras. La duración de la posición relativa del robot en cada galería se encuentra en la tabla 8.

Galería	Posición relativa(m)
Galería 17	Conocida de 5 a -4.5
Galería 16	Conocida de 5 a -1.8
Galería 15	Conocida de 5 a -5
Galería 14	Conocida de 5 a -1.2
Galería 13	Conocida de 5 a -4.5
Galería 12	Conocida de 5 a -5
Galería 11	Conocida de 5 a -4.8

Tabla 8: Posición relativa con patrones tomados de forma manual

Se puede apreciar como la incertidumbre es casi nula cuando se encuentra al lado de una galería.

Algunas de las imágenes tomadas justo cuando la posición relativa es nula, se encuentran en las figuras 29 y 30. Gracias a estas figuras se puede ver como cuadran los puntos de la traza con los patrones de puntos.

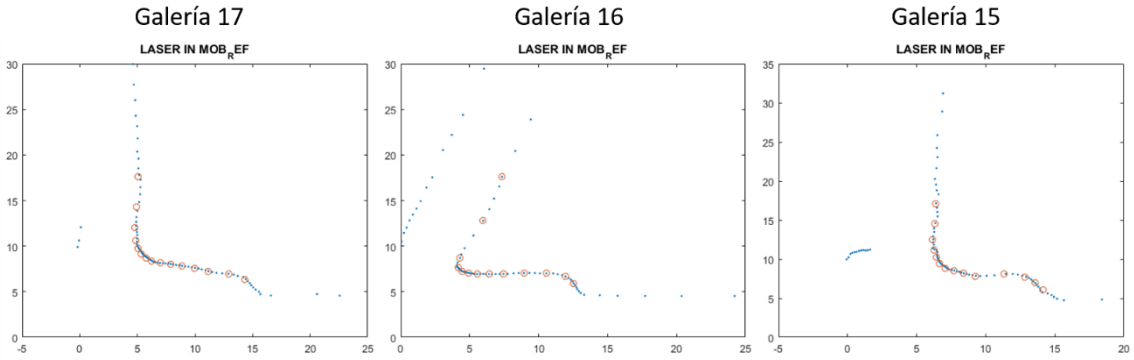

Figura 29: Posiciones relativas nulas del robot en las galerías utilizando patrón obtenido de forma manual

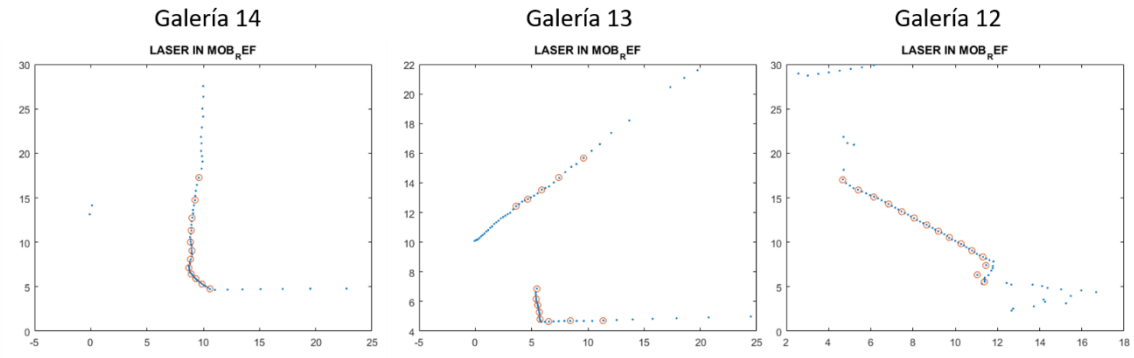

Figura 30: Posiciones relativas nulas del robot en otras galerías utilizando patrón obtenido de forma manual

Se puede apreciar como algunos puntos no coinciden exactamente, pero sí que son muy parecidos.

6.1.2 Patrones de puntos tomados de forma automática

Los resultados obtenidos utilizando patrones tomados de forma automática se pueden ver en la figura 31.

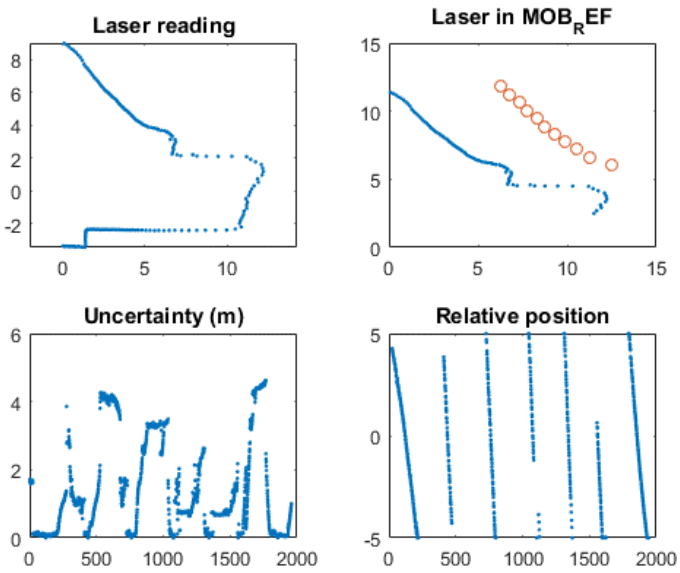

Figura 31: Resultados utilizando patrones tomados de forma automática

Se puede apreciar como los resultados son prácticamente iguales que los obtenidos con patrones tomados de forma manual. Esto quiere decir, que el método de automatizar la obtención de patrones funciona bien.

Como en el caso anterior, se pueden ver con más claridad las posiciones relativas del robot en la tabla 9.

Galería	Posición relativa(m)
Galería 17	Conocida de 4.3 a -5
Galería 16	Conocida de 4 a -4.3
Galería 15	Conocida de 5 a -5
Galería 14	Conocida de 5 a -2 y de -3.8 a -5
Galería 13	Conocida de 5 a -5
Galería 12	Conocida de 0.7 a -5
Galería 11	Conocida de 5 a -5

Tabla 9: Posición relativa con patrones tomados de forma automática

En comparación con la tabla 8, en algunas galerías la posición relativa del robot fue conocida durante más distancia como es el caso de la galería 16, pero otras fueron conocidas durante menos como la galería 12.

Las figuras 32 y 33 son realizadas en el momento en que la posición relativa es nula, y se puede ver como los puntos que conforman el patrón están más agrupados que en las figuras 29 y 30.

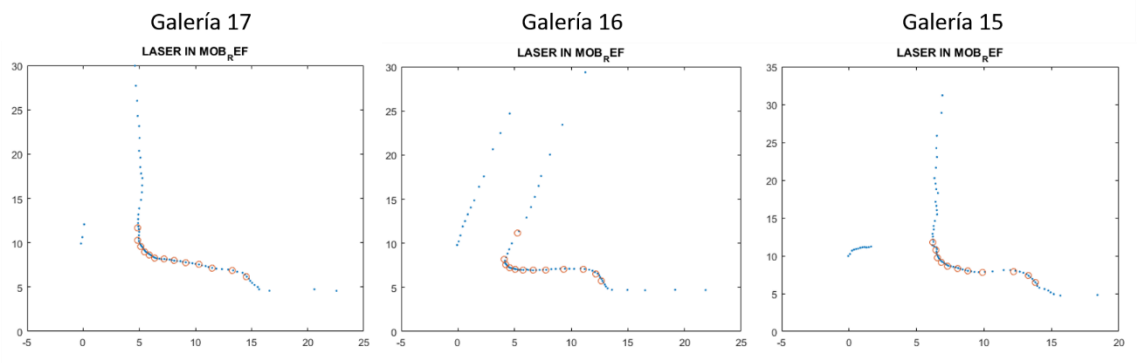

Figura 32: Posiciones relativas nulas del robot en otras galerías utilizando patrón obtenido de forma automática

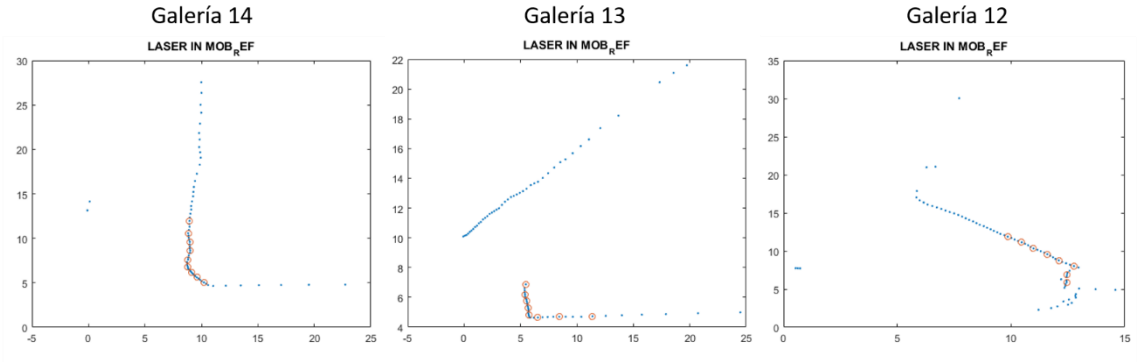

Figura 33: Posiciones relativas nulas del robot en otras galerías utilizando patrón obtenido de forma automática

Con estos resultados, se puede decir que ambas pruebas salieron favorables y, por lo tanto, el método de detección de galerías en MATLAB funciona a la perfección.

6.2 Trazas del archivo rosbag_ida.bag

Se utiliza un archivo de tipo bag que contiene unas trazas para asegurar el perfecto funcionamiento del método tanto en MATLAB como en ROS. A continuación, se analizan los resultados en ambos entornos.

6.2.1 Resultados en MATLAB

Para leer una traza en MATLAB, que está incluida en un archivo de tipo *bag*, y conocer el rango y el ángulo de los puntos de esta es necesario escribir lo siguiente en el terminal de MATLAB.

```
bag=rosbag('rosbag_ida.bag');
bSel = select(bag,'Topic','/laser/scan');

msgStructs = readMessages(bSel,1);
datos = cellfun(@(m) m.Ranges,msgStructs(1), 'UniformOutput',false);
rang=datos{1,1};
rango=rang';
angleMin = cellfun(@(m) m.AngleMin,msgStructs(1));
AngleMax = cellfun(@(m) m.AngleMax,msgStructs(1));
AngleIncrement = cellfun(@(m) m.AngleIncrement,msgStructs(1));
angle = (angleMin:AngleIncrement:AngleMax);
```

De esta forma se consigue sacar la información necesaria del archivo de tipo *bag*. Una vez que se conocen ya los rangos y los ángulos de los puntos de la traza del láser del archivo *bag*, se aplica el método de obtención automática de los puntos que conforman un patrón.

Una vez se obtienen los patrones de puntos, se aplica el método de detección de galerías.

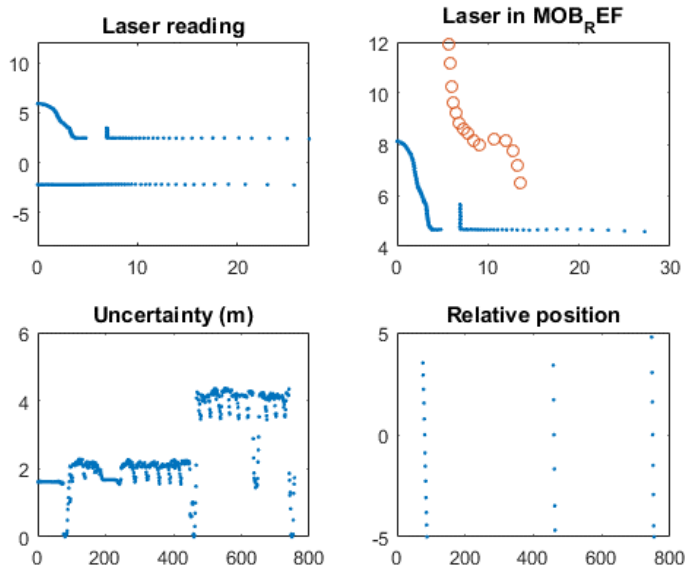

Figura 34: Resultados con trazas sacadas de archivo bag

En este caso, como se ve en la figura 34, la posición relativa se conoce durante menos muestras porque el robot va a una velocidad superior que en todos los casos anteriores y, por lo tanto, detecta la galería durante un periodo de tiempo más pequeño. Pero, gracias a la utilización de esta traza se puede garantizar que el robot detecta la galería a cualquier velocidad que este lleve y se puede realizar un perfecto emparejamiento, como se puede ver en la figura 35.

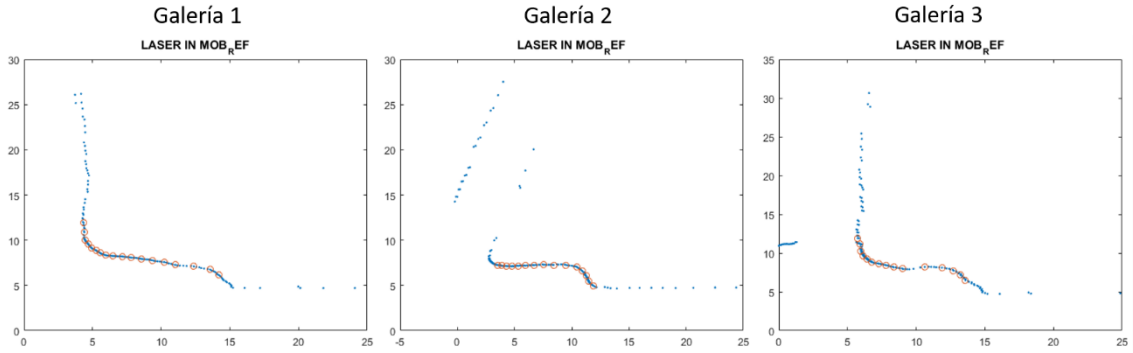

Figura 35: Comparación de trazas con el patrón de puntos en el momento que hay una posición relativa casi nula respecto a la galería en MATLAB

6.2.2 Resultados en ROS

Los patrones de puntos obtenidos en el subapartado anterior se guardan en el script scan.py.

La información del archivo *bag* en ROS se obtiene mediante el comando c.11

Rosbag play –pause rosbag_ida.bag –rate 1 (c.11)

Cada vez que se presione a la barra espaciadora se pone la traza en pausa o en marcha. El rate 1 significa que va a tiempo real. También, se puede iniciar en un tiempo concreto del archivo añadiendo –start a un tiempo en c.11.

Inicialmente se debe poner el comando roscore, para cargar el ROS Master.

Finalmente, se realiza un roslaunch del tfg.launch y miramos la información que se publica en el *topic* /informacionTunel.

La figura 36 muestra el emparejamiento de trazas con el patrón de puntos en el momento que hay una posición relativa casi nula respecto a la galería.

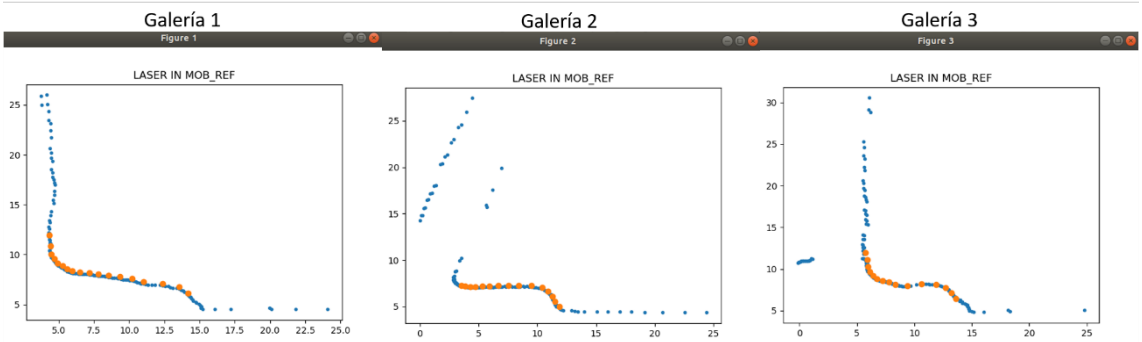

Figura 36: Emparejamiento de trazas con el patrón de puntos en el momento que hay una posición relativa casi nula respecto a la galería en ROS

Se puede ver como entre la figura 35 y la figura 36 no hay apenas diferencias, por lo tanto, funcionan ambos métodos bien.

6.3 Túneles sintéticos en gazebo

Para poder garantizar el perfecto funcionamiento del método en ROS, se realizan túneles sintéticos en el simulador Gazebo **[15]**.

Gazebo es un simulador de entornos en 3D que permite evaluar cómo se va a comportar un robot creando algoritmos rápidamente. También, se puede diseñar robots o entrenar una inteligencia artificial en escenarios realistas. Permite simular con gran eficiencia en entornos exteriores e interiores. Estos entornos o mundos virtuales los puede crear uno mismo gracias a que se pueden importar modelos 3D ya creados, usando sencillas herramientas de CAD o en el propio *model editor* que posee gazebo. También, posee gráficos de alta calidad y es gratuito.

Una de las ventajas de este simulador es que se puede sincronizar con ROS. Esto permite, a través de códigos, dar órdenes al robot o recibir la información de los láseres que posea el robot suscribiéndote a dicho *topic*.

Para realizar estos túneles propios se utilizan modelos 3D de entradas, intersecciones o secciones curvas o rectas de túneles. Estos modelos 3D, junto a un túnel de prueba ya creado y su lanzador, y un robot, fueron proporcionados por el departamento de robótica. El lanzador del túnel y estos modelos 3D se encuentran dentro de la carpeta subt_gazebo.

El robot, que se muestra en la figura 37, para estas simulaciones tiene un láser con un alcance de 30m, por lo tanto, se puede realizar correctamente la automatización de los puntos clave. También posee un ángulo máximo de $\pi/2$ y un ángulo mínimo de $-\pi/2$, con el que se pueden ver perfectamente las dos paredes.

Figura 37: Robot empleado en las simulaciones

Los mundos que abre el simulador se escriben en lenguaje de programación xml y se pueden ver en Anexos.

El manejo del robot en los túneles se puede realizar con teclas del teclado en un terminal **[16]**. Esto se consigue instalando ros-kinetic-teleop-twist-keyboard en el ordenador. Una vez instalado se escribe en el terminal el comando c.12.

rosrun teleop_twist_keyboard teleop_twist_keyboard.py cmd_vel:=/sim_p3at/cmd_vel (c.12)

La última parte de c.12, cmd_vel:=/sim_p3at/cmd_vel, se introduce porque se quiere que la información que se envía desde el terminal sea publicada al *topic* que se encarga de enviar información del movimiento del robot. Los diferentes movimientos que le puedes comunicar al robot desde terminal son:

Moving around:
u i o
j k l
m , .

q/z : increase/decrease max speeds by 10%
w/x : increase/decrease only linear speed by 10%
e/c : increase/decrease only angular speed by 10%
anything else : stop
CTRL-C to quit

6.3.1 Túnel propio 1

Se realiza un primer túnel con los modelos 3D que ha proporcionado en departamento.

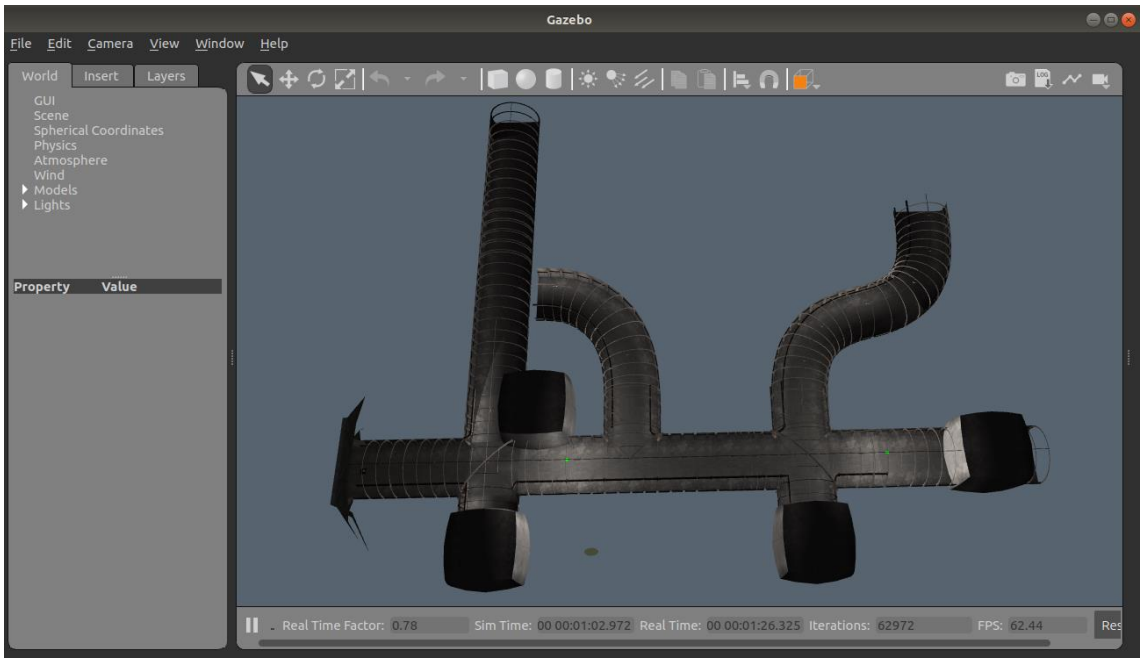

Figura 38: Primer túnel sintético en gazebo

La figura 38 muestra el túnel creado. Este túnel posee tres galerías, cada una de ellas con unas características diferentes para ver si el método de detección de galerías funciona bien en ellas.

Inicialmente, se recorre con el robot, gracias a los movimientos vía teclado, todo el túnel. Esto se realiza para guardar en un archivo de tipo *bag* toda la información que publica el láser frontal en su *topic*. El comando c.13 inicializa el túnel y el comando c.14 sirve para grabar en un archivo bag la información de un *topic* en específico.

roslaunch subt_gazebo tunnel.launch (c.13)

rosbag record /laser_front/scan (c.14)

Después, se carga en MATLAB dicho archivo *bag*, como se ha explicado en el apartado 6.2, para aplicar el método que saca automáticamente los patrones de puntos de cada galería. Se escriben los diferentes patrones en el archivo sacan.py.

Una vez realizado esto, se escriben los comandos que hay en la figura 39 en diferentes terminales.

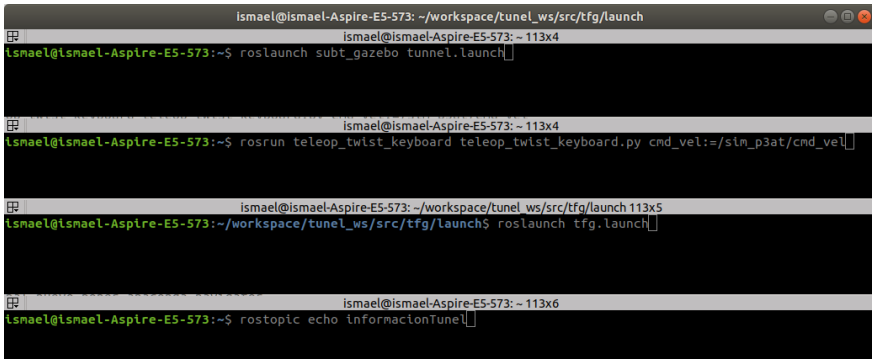

Figura 39: Comandos para realizar la detección

Los dos primeros comandos se explican antes. El tercer comando es como se lanza el archivo donde se encuentra el método de detección de galerías, y con el cuarto comando se lee la información que es publicada en el *topic* /informacionTunel.

Al introducir los puntos que conforman el patrón de cada una de las tres galerías en el archivo scan.py, se está intentando emparejar el patrón de puntos de la primera galería con los puntos de la traza que está tomando el láser del robot en el túnel sintetizado en 3D en tiempo real.

El método también funciona a la perfección en ROS. Esto se sabe gracias a los mensajes que se leen del *topic* donde se publica la información del túnel, como la posición relativa del robot respecto a la galería o si se está detectando o no la galería, entro otras cosas. También, gracias a la importación de la librería matplotlib.pyplot como plt **[17]**, se pueden realizar figuras de cuando el robot se encuentra en la posición relativa nula respecto a la galería como en MATLAB. Estas figuras indican que se está realizando un buen emparejamiento entre patrón y traza del láser en tiempo real.

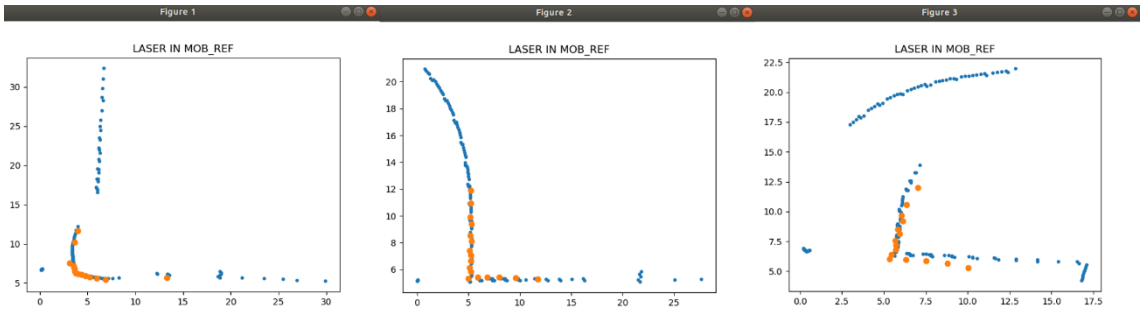

Figura 40: Emparejamiento de las trazas del láser con los patrones de puntos

La figura 40 muestra el emparejamiento que hay entre el patrón de cada galería y la respectiva traza del láser.

6.3.2 Túnel propio 2

En la figura 41, se muestra otro túnel sintetizado en el simulador Gazebo.

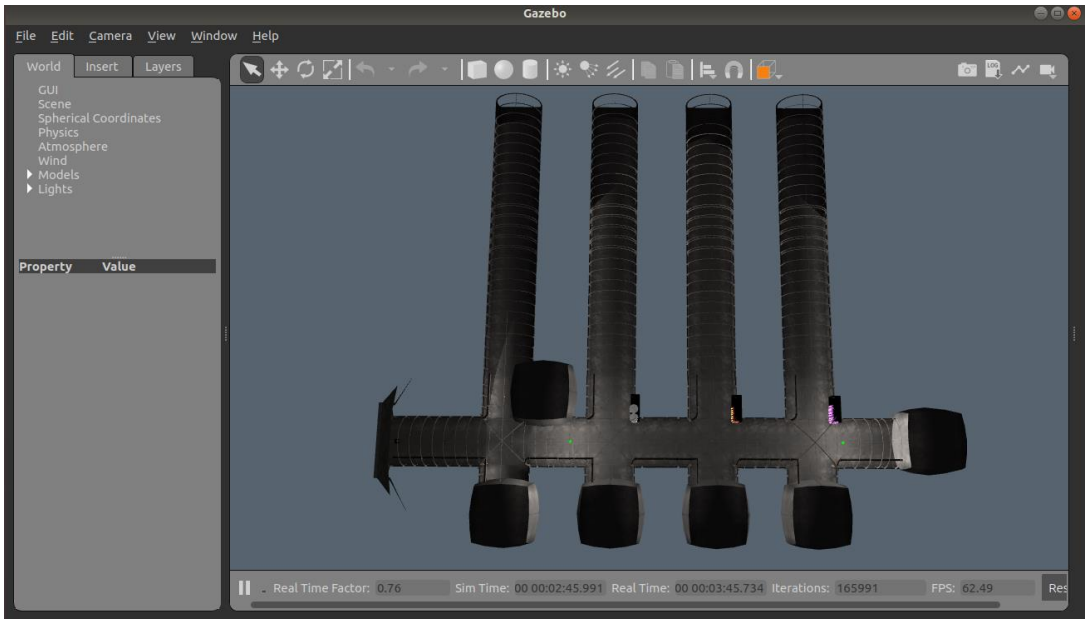

Figura 41: Segundo túnel sintético en gazebo

Este túnel está constituido por cuatro galerías idénticas, pero en cada una de ellas se pone un bloque en la parte derecha de esta. Estos bloques hacen que las trazas del láser no sean iguales en las cuatro galerías. El primer bloque esta junto a los modelados 3D de las secciones de los túneles. Los otros tres bloques se han creado en la pantalla *model editor* de Gazebo con figuras geométricas. Hay un primer plano de estos bloques creados en la figura 42.

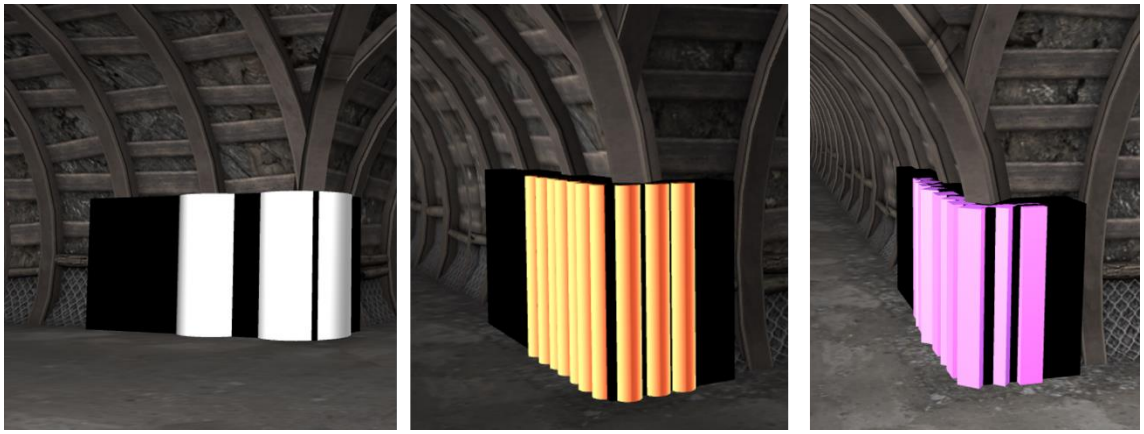

Figura 42: Bloques creados en el model editor de Gazebo

Posteriormente, se hace el mismo procedimiento que en el apartado 6.3.1.

Las figuras 43 y 44 son obtenidas cuando la posición relativa del robot es nula respecto a la ubicación de la galería.

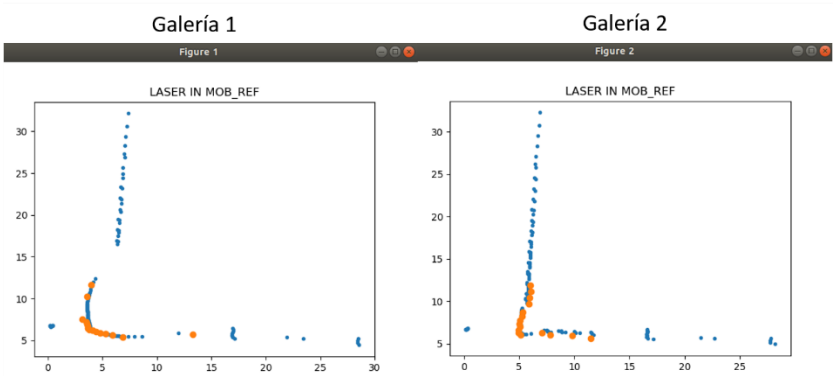

Figura 43: Emparejamientos galerías 1 y 2

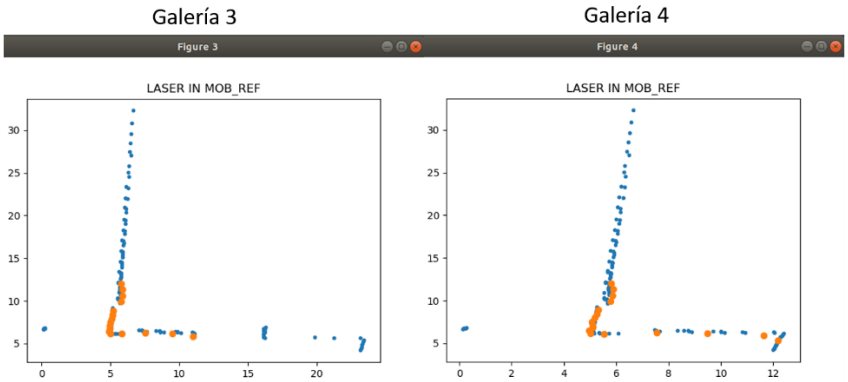

Figura 44: Emparejamientos galerías 3 y 4

Con todas estas pruebas, se puede concluir que el método funciona tanto en MATLAB como en ROS.

Capítulo 7

Conclusiones

Gracias al conjunto de elementos estructurales para mejorar la seguridad en los túneles, se puede llegar a tener un robot localizado en un túnel en su distancia longitudinal.

Es más favorable detectar las galerías debido a que cada una tiene una forma distinta. También, se pueden detectar apartaderos y refugios, pero estos ya poseen la misma forma. Sin un mapa detallado es difícil diferenciar cuál de los apartaderos o refugios se ha detectado.

La transformada de Hough es una herramienta para saber la posición y ángulo del robot respecto a las paredes del túnel cuando se detecta gran parte del muro del túnel, algo que ocurre en los túneles reales debido a que los elementos estructurales están separados entre ellos. En los túneles sintéticos la distancia entre galerías es pequeña y no queda una línea recta del muro tan detalla como en los reales y produce un pequeño giro de más a aplicar en los puntos de la traza. Este giro no es un inconveniente porque el emparejamiento se sigue realizando de manera correcta.

Gracias a Gazebo se han podido realizar entornos de simulación con un robot, y gracias a ROS se han podido probar la implementación del método de detección de galerías en este entorno. Poder suscribirte al *topic* donde el láser guarda la información, permite realizar emparejamientos entre los puntos que este capta y los puntos del patrón en tiempo real.

La utilización del *model editor* de Gazebo es sencilla y puedes crear los modelos 3D necesarios. También, se pueden incluir en los mundos modelos 3D creados con otros programas de modelado.

La detección de las galerías es independiente de la velocidad del robot. Esto ha quedado reflejado al realizar las pruebas con una traza de mayor velocidad.

La métrica más factible para realizar la comparación y emparejamiento de los puntos del patrón y de la traza es la Nearest y el modelo para calcular el error de posición es el error medio, M.

La utilización del archivo *bag* para el almacenamiento de la información de la traza realizada por el láser en el túnel es la mejor opción, ya que este tipo de archivo puede ser utilizado tanto en MATLAB como en ROS, algo que no ocurre con los archivos .mat.

Capítulo 8

Bibliografía

[1] Unión Europea. Directiva (UE) 2004/54/CE del Parlamento Europeo y del Consejo, de 29 de abril de 2004, sobre requisitos mínimos de seguridad para túneles de la red transeuropea de carreteras. Diario Oficial de la Unión Europea L 167, 30 de abril de 2004, pp. 1-21.

[2] «MathWorks - Creadores de MATLAB y Simulink - MATLAB y Simulink,» Disponible en: https://es.mathworks.com.

[3] P. Hough, «Method and means for recognizing complex patterns». Patente 3069654, 18 Diciembre 1962.

[4] «Hough transform,» Wikipedia, La enciclopedia libre, 1 Agosto 2019. Disponible en: https://en.wikipedia.org/wiki/Hough_transform.

[5] «Documentation - ROS Wiki,» Disponible en: http://wiki.ros.org.

[6] «NumPy v1.17 Manual,» Disponible en: https://docs.scipy.org/doc/numpy/index.html.

[7] «9.2. math — Mathematical functions — Python 2.7.16 documentation,» Disponible en: https://docs.python.org/2.7/library/math.html.

[8] «Python range for Float Numbers with Examples,» Disponible en: https://pynative.com/python-range-for-float-numbers.

[9] «Optimization and root finding (scipy.optimize) — SciPy v0.16.1 Reference Guide,» Disponible en: https://docs.scipy.org/doc/scipy-0.16.0/reference/optimize.html.

[10] «13. Enumerate — Python Tips 0.1 documentation,» Disponible en: http://book.pythontips.com/en/latest/enumerate.html.

[11] «Python Global, Local and Nonlocal variables (With Examples),» Disponible en: https://www.programiz.com/python-programming/global-local-nonlocal-variables.

[12] «How to implement a switch-case statement in Python,» Disponible en: https://jaxenter.com/implement-switch-case-statement-python-138315.html.

[13] «Diccionarios y listas en Python,» Disponible en: https://www.tutorialpython.com/listas-en-python.

[14] «scipy.spatial.cKDTree — SciPy v1.3.0 Reference Guide,» Disponible en: https://docs.scipy.org/doc/scipy/reference/generated/scipy.spatial.cKDTree.html#scipy.spatial.cKDTree.

[15] «Gazebo,» Disponible en: http://gazebosim.org.

[16] «teleop_twist_keyboard - ROS Wiki,» Disponible en: http://wiki.ros.org/teleop_twist_keyboard.

[17] «Matplotlib: Python plotting — Matplotlib 2.0.2 documentation,» Disponible en: https://matplotlib.org/2.0.2/index.html.

Capítulo 8

Anexos

8.1 Códigos de MATLAB

Tunnelwall.m

```
%Este archivo contiene el método de detección de galerías utilizando
trazas guardadas en archivos .mat o archivo .bag. SI se utiliza
archivo Ros antes hay que abrir este con bag=rosbag('X.bag') y
seleccionar el topic que se quiere leer con
bSel=select(bag,'Topic','/laser_front/scan').
G = []; %Lista donde se guardan todas los objetos de la clase Gallery.
%La clase Gallery tiene como propiedad xp (puntos del patrón en la
cooordenada X), yp (puntos del patrón en la coordenada Y), delta_x (punto
de referencia del posicionamiento de la galería)y name (el número de la
galería).
aux = Gallery ; %Se crea variable de la clase Gallery.
aux.xp = [ 5.99  6.18  6.11  6.04 7.23 10.57 13.49 15.38 16.36 17.59] ;
%Ejemplo de puntos de patrón en coordenada X que se añaden a la propiedad
xp del objeto de la clase Gallery.
aux.yp = [26.28 19.56 14.01 10.51 8.55  7.83  7.13  6.61  5.38  4.66] ;
%Ejemplo de puntos de patrón en coordenada Y en la propiedad yp.
%aux.xp = [6.18  6.11  6.04 7.23 10.57 13.49 15.38 16.36 17.59 6.37 8.81
] ; %Otro patrón diferente
%aux.yp = [15.56 14.01 10.51 8.55  7.83  7.13  6.61  5.38  4.66 9.56
8.205 ] ;
aux.delta_x = 6.3 ; %Se añade la propiedad del posicionamiento de la
galería al objeto.
aux.name = 'Galeria 17' ;  %También se añade el nombre.
G = [G aux] ;  %Se añade el objeto creado a la lista.
galeria =1 ; %Se escoge el patrón de puntos deseado.
px = G(galeria).xp ;  %Se crean las variables necesarias tomando los
valores de las propiedades del objeto que corresponde a la galería a
detectar.
py = G(galeria).yp ;
th_1=0; rh_1=0; %Por si la transformada de Hough falla, se guarda el
valor anterior.
errores_min = [] ;  errores_pos = [] ;  deltax = [] ; %Listas para
guardar la información.
global Pattern_X; global Pattern_Y ; global Laser_X ; global Laser_Y ;
global Metric ; global Method ; %Se crean las variables globales
necesarias para aplicar la función PatternDistance.
figure (1) ;
for k = 7350:44900 %Ciclo en el que se van leyendo todos los valores de
los archivos angles2.mat y ranges2.mat o de uno de los archivos .bag.
      msgStructs = readMessages(bSel,k); %Necesario solo cuando se
utilizan archivos .bag, en el cual se lee el mensaje y se puede obtener
los valores de los vectores de rangos y ángulos.
      angleMin = cellfun(@(m) m.AngleMin,msgStructs); %Necesario solo
cuando se utilizan archivos .bag, nos da información del ángulo mínimo.
      AngleMax = cellfun(@(m) m.AngleMax,msgStructs); %Solo par archivos
.bag, nos da información del ángulo máximo.
```

```
      AngleIncrement = cellfun(@(m) m.AngleIncrement,msgStructs);
      %Necesario solo cuando se utilizan archivos .bag y nos proporciona
el incremento de los ángulos.
      angles = (angleMin:AngleIncrement:AngleMax); %Necesario solo
cuando se utilizan archivos .bag. Se crea la lista con los ángulos de
los puntos del láser.
      datos = cellfun(@(m) m.Ranges,msgStructs, 'UniformOutput',false);
      %Necesario solo cuando se utilizan archivos .bag, proporciona el
rango de los puntos del láser.
      rang=datos{1,1}; %Necesario solo cuando se utilizan archivos .bag.
      ranges=rang'; %Necesario solo cuando se utilizan archivos .bag.
      % Se filtran los puntos con un rango mayor a 30m.
      [a  r]  =  R_filter  (angles,  ranges);  %Necesario  solo  cuando  se
utilizan archivos .bag.
      %[a r] = R_filter (angles, ranges (k,:)) ; %Necesario solo cuando
se utilizan archivos .mat.
      [lx ly] = Polar2Cart(a, r) ; %Datos a coordenadas cartesianas para
hacer el emparejamiento con el patrón.
      I = find (ly<=8) ; %Se guardan los índices de posición de los
      puntos que cumplen la condición.
      lx_HT=lx(I); %Se guardan en una nueva lista los puntos que cumplían
la condición.
      ly_HT=ly(I);
      rho,theta]  =  houghTransform(lx_HT,  ly_HT)  ;  %Se  realiza  la
      transformada de Hough.
      if ((theta>=-0.05) && (theta<=0.05)) %Si el valor es próximo a
cero es que se ha realizado la transformada mal y se le asigna el
valor anterior bueno.
          theta=th_1;
          rho=rh_1;
      end
      figure (1) ;
      subplot (2,2,1) ; %Se muestra en una primera gráfica la lectura
      del laser.
      plot_laser (a, r) ;
      title ('Laser reading') ;
      % Matriz de rotación para el láser
      alfa = -(pi/2 + theta) ; %Ángulo que se rota en eje Z.
      R = [cos(alfa)  -sin(alfa) ; sin(alfa) cos(alfa)] ; %Matriz de
rotación en el eje Z 2x2.
      % Se rota la traza del láser con la matriz de rotación.
      laser_rotado = R*[lx ; ly] ;
      lx = fliplr(laser_rotado(1,:)) ; % Invierte las filas por las
columnas y viceversa, para crear la nueva lista de los puntos
      ly = fliplr(laser_rotado(2,:)) ;
      [lx, ly] = Y_filter (lx, ly) ; %Se eliminan los puntos del láser
que tienen una coordenada en Y negativa.
      ly = ly + rho ;
      figure (1) ;
      subplot (2,2,2) ; %Se muestra en una segunda gráfica el láser en
referencia del robot junto con el patrón de puntos.
      plot (lx, ly, '.') ;
      title ('Laser in MOB_REF') ;
      hold on ; %Para poder mostrar también los puntos del patrón.
      plot (px, py, 'o') ;
      hold off
```

```
      Pattern_X=px; Pattern_Y=py; Laser_X=lx; Laser_Y=ly; %Se asigna un
      valor a las variables globales.
      Metric = 'Nearest'; %Se elige la métrica que sea mejor para la
      detección.
      %Metric = 'X';
      %Metric = 'Polar';
      %Method = 'q'; %Se elige el método para calcular el error de
posición.
      Method = 'm';
      %Method = 'h';
      [xmin err_min] = fminbnd (@XPatternDistance, -5, 5) ; %Se busca la
distancia mínima utilizando la función XPatternDistance en un intervalo
entre -5 y 5m.
      err_pos = XPatternDistance(0) ; %Se calcula el error de posición
respecto al punto de referencia de la galería.
      errores_min = [errores_min err_min] ; %Se van guardando los errores
      errores_pos = [errores_pos err_pos] ;
      deltax = [deltax xmin] ; %Se van guardando las distancias mínimas.
      figure (1) ;
      subplot (2,2,3); %Se muestra en una tercera gráfica la
incertidumbre.
      plot (errores_min, '.') ;
      title ('Uncertainty (m)') ;
      I = find (errores_min < 0.5) ; %Si la incertidumbre es menor a
0.5, se muestra la posición relativa.
      figure (1) ;
      subplot (2,2,4) ; %Se muestra en una cuarta gráfica la posición
      relativa.
      plot (I,deltax(I), '.') ;
      title ('Relative position') ;
      pause (0.1) , %Se para durante un momento el método para que se
pueda actualizar la figura.
      th_1=theta; %Se actualiza el valor de la tita anterior con el
actual por si falla la transformada de Hough.
      rh_1=rho;
end; %Final de bucle for y del método.
```

PatternDistance.m

```
function Distance = PatternDistance(px, py, lx, ly, dist, method)
%Función en la que están implementadas todas las métricas para el
emparejamiento del patrón con la traza y métodos para sacar el error
de posición. En los parámetros de la función ya eliges que métrica y
que método quieres utilizar.
switch (dist) %Switch en el que se escoge el tipo de métrica que se
aplica para el emparejamiento.
    case 'Polar' %Métrica Polar
        %Se pasan todos los puntos del láser y del patrón a
coordenadas polares.
        [ldist, lang] = Cart2Polar (lx, ly) ;
        [pdist, pang] = Cart2Polar (px, py) ;
        d = [] ; %Lista donde se guardan todas las distancias.
        % Se compara cada uno de los puntos del patrón con todos los
puntos del láser y se van añadiendo las distancias a la lista.
        for j = 1:length(pang)
            ind = find (lang<pang(j),1) ;
            if length(ind) == 0
```

```
                d = [d (80.0-pdist(j))] ;
            else
                if ind > 1 %Por si toma dos índices de posición.
                    ind = ind -1 ;
                end ;
                d = [d abs(ldist(ind)-pdist(j))] ;
            end ;
        end ;
    case 'Nearest' %Métrica Nearest
        %Se utiliza la función dsearch de MATLAB que te devuelve la
distancia mínima a cada punto del patrón.
        [k, d] = dsearchn ([lx' ly'], [px' py']) ;
    case 'X' %Métrica X
        d = [] ;
        % Se compara cada uno de los puntos del patrón.
        for j = 1:length(py)
            [aux, I] = min(abs(ly - py(j))) ;
            %Si la distancia entre las coordenadas Y es menor a 0.5m,
se añade a la lista de distancias la diferencia entre las coordenadas
X de los puntos del patrón y la traza.
            if aux < 0.5
                d = [d abs(px(j) - lx(I))] ;
            else
                d = [d 80.0] ; %Rango máximo del láser utilizado para
las trazas.
            end ;
        end ;
end

switch (method) %Switch en el que se escoge el tipo de método para
calcular el error de posición.
    case 'q' %Método que devuelve el error cuadrático medio.
        Distance = sqrt(sum(d.^2)/length(d))  ;
    case 'm' %Método que devuelve el error medio.
        Distance = mean (d) ;
    case 'h' %Método que devuelve el error máximo.
        Distance = max (d) ;
end
end
```

XPatternDistance.m

```
function d = XPatternDistance(x)
   %Función que suma el valor del parámetro a la coordenada X de los
puntos que conforman el patrón. Se utiliza para saber qué valor del
intervalo [-5,5] nos da la menor distancia. Las variables son globales
para que PatternDistance reconozca sus valores.
   global Pattern_X ; global Pattern_Y ; global Laser_X ;
   global Laser_Y ; global Metric ; global Method ;
   d = PatternDistance(Pattern_X + x, Pattern_Y, Laser_X, Laser_Y,
Metric, Method) ; %Se aplica la función PatternDistance, que devuelve
el error de posición mínimo.
end
```

houghTransform.m

```
function [rho,theta] = houghTransform(lx, ly)
    %Esta función realiza la transformada de Hough y nos devuelve la
coordenada radial y angular del robot frente a los muros del túnel.
    thetaSampleFrequency = pi/180 ; % 1 grado
     rho = (-30.0:0.2:30.0); %Suelen ser los rangos típicos que poseen
los láseres.
    theta = (-pi:thetaSampleFrequency:0); %Perímetro de alcance del
láser con un incremento de 1 grado.
    accumulator = lx'*cos(theta) + ly'*sin(theta) ; %Se aplica esta
operación para cada una de las titas de la lista o array y se guardan
todos los resultados en la variable accumulator. Los resultados son
las posibles coordenadas radiales.
     numThetas = numel(theta); %Longitud de la lista de titas.
    houghSpace = zeros(numel(rho),numThetas); %Se crea una matriz de
ceros con tantas filas como la longitud de la lista de rangos y tantas
columnas como el valor de numThetas.
    for i = (1:numThetas) %Para la longitud de la lista de titas.
        houghSpace(:,i) = hist(accumulator(:,i),rho); %Se realiza un
histograma para saber cuántas veces se repite cada valor de la lista
accumulator. Se aplica a una columna todos los valores del histograma.
El valor que más se repita es la coordenada radial buena.

    end
    I = find (rho >= 0.0) ; %Se guardan los índices de la posición en
la lista de los valores con rango mayor a cero para posteriormente
eliminarlos de la matriz houghSpace.
    rho = rho(I) ;
    houghSpace = houghSpace(I,:) ;
    [Y,I] = max (houghSpace) ; %Se proporciona el valor y el índice de
posición de la columna donde se encuentra el máximo.
    [M,J] = max (Y) ; %Se proporciona el índice de posición del  valor
máximo de la columna.
    rho = rho(I(J)) ; %Se proporciona el índice de posición del  valor
máximo de la columna.
    theta = theta(J) ;
end
```

tunnelwall_PuntosClavesAutomatizados.m

```
%Este archivo realiza la obtención de los puntos que conforman el
patrón de una galería de forma automática. Se obtiene un patrón, con
varias condiciones explicadas en el estudio sobre la localización de
los puntos clave, para cada una de las galerías que conforman el
túnel.
%Declaración de las variables que se utilizan.
True=1; False=0; ContadorHayGaleria=0; x_clave=[]; y_clave=[];
UnicaVez=True;
G=[]; th_1=0; rh_1=0;
k=6500; %Punto de comienzo del bucle while.
figure (1) ;
 while k <=52000 %Se realiza el bucle hasta que se deja de cumplir la
condición.
    msgStructs = readMessages(bSel,k); %Uso con datos de archivo .bag
    angleMin = cellfun(@(m) m.AngleMin,msgStructs); %Uso con datos de
archivo .bag
```

```
    AngleMax = cellfun(@(m) m.AngleMax,msgStructs); %Uso con datos de
archivo .bag
    AngleIncrement = cellfun(@(m) m.AngleIncrement,msgStructs); %Uso
con datos de archivo .bag
    angles = (angleMin:AngleIncrement:AngleMax); %Uso con datos de
archivo .bag
    datos = cellfun(@(m) m.Ranges,msgStructs, 'UniformOutput',false);
%Uso con datos de archivo .bag. La salida uniforme se pone a falso
para poder leer el archivo de datos.
    rang=datos{1,1}; %Uso con datos de archivo .bag
    ranges=rang'; %Uso con datos de archivo .bag
    hayPntIzq=False; hayGaleria=False;
    %La explicación de esta parte del método se encuentra en el
archivo tunnelwall.m.
    %[a, r] = R_filter (angles, ranges (k,:));  %Archivos .mat
    [a, r] = R_filter (angles, ranges);  %Archivos .bag
    [lx, ly] = Polar2Cart(a, r) ;
    I = find (ly<=8) ;
    lx_HT=lx(I);
    ly_HT=ly(I);
    [rho,theta] = houghTransform(lx_HT, ly_HT) ;
    if ((theta>=-0.05) && (theta<=0.05))
        theta=th_1;
        rho=rh_1;
    end
    figure (1) ;
    subplot (2,1,1) ;
    plot_laser (a, r) ;
    title ('Laser reading') ;
    % Matriz de rotación para el láser
    alfa = -(pi/2 + theta) ;
    R = [cos(alfa)  -sin(alfa) ; sin(alfa) cos(alfa)] ;
    % Se rota el láser
    laser_rotado = R*[lx ; ly] ;
    lx = fliplr(laser_rotado(1,:)) ;
    ly = fliplr(laser_rotado(2,:)) ;
    [lx, ly] = Y_filter (lx, ly) ;
    ly = ly + rho ;
    %Hasta aquí está todo el método explicado en tunnelwall.m.
    %Obtención de forma automática de punto claves
    [r_auto, tita_auto]=Cart2Polar(lx,ly) ;
    for i=1:length(r_auto)
        if ((r_auto(i) >= 12) && (tita_auto(i) >= pi/4)) %Condición
para saber si tenemos una galería alrededor.
            hayGaleria=True;
        end
    end
    for i=1:length(lx)
        if ((lx(i)<0.5) && (ly(i)<10.0)) %Condición para saber si hay
un punto justo a la izquierda del robot. Si esto se cumple no se puede
tomar aún el patrón de puntos.
            hayPntIzq=True;
        end
    end
    if hayGaleria==True
        %Se realiza un contador para no tomar la traza justo en el
primer momento que se detecta que hay galería.
        ContadorHayGaleria=ContadorHayGaleria+1 ;
    else
```

```
        for i=1:length(r_auto) %Si después de detectar una galería se
detecta de nuevo el muro del túnel, las variables vuelven a estar en
su valor predeterminado.
            if ((r_auto(i) <= 5.5) && (tita_auto(i) >= pi/2.1))
            UnicaVez=True;
            ContadorHayGaleria=0;
            end
        end

    end
    if ((ContadorHayGaleria>=2) && (UnicaVez==True) &&
(hayPntIzq==False)) %Condiciones para realizar el patrón una vez por
galería en el punto de referencia de la galería.
        I = find (lx >= 3.5) ; %Primera filtración para eliminar
puntos alejados en la coordenada X.
        x_clave=lx(I);
        y_clave=ly(I);
        I = find (y_clave < 12.0) ; %Segunda filtración para eliminar
puntos muy alejados en la coordenada Y.
         x_clave=x_clave(I);
        y_clave=y_clave(I);
        [r_auto, tita_auto]=Cart2Polar(x_clave, y_clave); %Se cambian
los puntos a coordenadas polares para realizar otras filtraciones.
        I = find (r_auto<=20) ; %Tercera filtración para eliminar
puntos con una coordenada radial alta.
        r_clave=r_auto(I);
        a_clave=tita_auto(I);
        I = find (a_clave>=(pi/8.0)); %Cuarta filtración para eliminar
datos con una coordenada angular menor a 22.5 grados.
        r_clave=r_clave(I);
        a_clave=a_clave(I);
        [x_clave, y_clave]=Polar2Cart(a_clave,r_clave);
        if length(x_clave)<=60 %Ultima filtración para reducir el
tamaño de los puntos que conforman el patrón. Si son más de 60 puntos
se eliminan cinco de cada seis puntos, sino tres de cada cuatro.
            x_clave=x_clave(1:4:end);
            y_clave=y_clave(1:4:end);
        else
            x_clave=x_clave(1:6:end);
            y_clave=y_clave(1:6:end);
        end
        UnicaVez=False;
        ContadorHayGaleria=0;
        galeria = Gallery ; %Se guardan los puntos resultantes en un
objeto de la clase Gallery.
        galeria.xp=x_clave;
        galeria.yp=y_clave;
        G = [G galeria] ; %Se añade el objeto a la lista de galerías.
    end
    figure (1) ;
    subplot (2,1,2) ; %Grafica que muestra los laser respecto a la
referencia del robot.
    plot (lx, ly, '.') ;
    title ('Laser in MOB_REF') ;
    mostrar=['punto: ', num2str(k)]; %Información para ver por
terminal en que parte del túnel está.
    disp (mostrar)
    %Se muestra por pantalla si hay galería.
    mostrar=['Hay galeria: ', num2str(hayGaleria)];
    disp (mostrar)
```

```
    %Se muestra por pantalla si se han tomado ya una vez el patrón de
puntos de la galería.
    mostrar=['Unica Vez: ', num2str(UnicaVez)];
    disp (mostrar)
    %Se muestra por pantalla si hay punto la izquierda del robot.
    mostrar=['Punto izquierda: ', num2str(hayPntIzq)];
    disp (mostrar)
    %Se muestra por pantalla la longitud de la lista de objetos de la
clase Gallery para saber de cuantas galerías del túnel se ha realizado
el patrón.
    mostrar=['Longitud G: ', num2str(length(G))];
    disp (mostrar)
    th_1=theta; %Se asignan los nuevos valores a las variables del
valor anterior.
    rh_1=rho;
    k=k+1; %Se incrementa en uno el valor, para que cuando se cumpla
la condición se pare el bucle.
    pause (0.1) , %Pausa para que se muestre la figura con las dos
gráficas.
end ; %Final del método.
```

8.2 Códigos de ROS

Scan.py

```
#! /usr/bin/env python
#Archivo de tipo py donde se encuentra la parte principal del método de
detección de galerías. El main es la función callback, que se ejecuta
cada vez que se recibe un mensaje del topic al que se está suscrito.
import rospy, deftfg, numpy as np, math, matplotlib as plt  #Diferentes
imports para implementar el método.
from scipy import optimize
from scipy.spatial import cKDTree
from sensor_msgs.msg import LaserScan #Se importa para recibir mensajes
de tipos LaserScan.
from std_msgs.msg import String #Se importa para poder realizar mensajes
de tipo String.
errores_min=[]; errores_pos=[]; deltax=[]  #Creacion de variables
Pattern_X="";   Patterns_Y="";   Laser_X="";   Laser_Y="";   Metric="";
Method=""
#Patrones de puntos en coordenadas X e Y de las diferentes galerías del
archivo bag.
posicionPC_X=[[4.3487430,  4.3989511,  4.4705315,  4.7028122,  4.9348817,
5.2644305,  5.5719280,  5.9561338,  6.5055962,  7.1630182,  7.8147097,
8.5398445,  9.3321934,  10.201313,  11.009238,  12.375752,  13.560403,
14.195428],[3.5332327,  3.8909805,  4.2971873,  4.7514639,  5.3169432,
5.9291830,  6.6845808,  7.5359530,  8.4173832,  9.5093479,  10.431603,
10.912955,  11.261127,  11.515915,  11.906802],[5.7351832,  5.9278045,
5.9720221,  6.1381550,  6.4181070,  6.7544837,  7.2648706,  7.8182564,
8.3609676,  9.411558,  10.627043,  11.875779,  12.728794,  13.235179,
13.590467]]
posicionPC_Y=[[11.925907,  10.873219,  10.040213,  9.5722055,  9.1488104,
8.8739033,  8.5751591,  8.3517714,  8.2552290,  8.1963625,  8.0719185,
7.9362869,  7.7729182,  7.5755119,  7.2731438,  7.1044054,  6.7458301,
6.1393166],[7.2459760,  7.1802278,  7.1433468,  7.1202836,  7.1581273,
7.1751757,  7.2371912,  7.2830629,  7.2576728,  7.2562079,  7.0643353,
```

```
6.6091180,  6.0775161,  5.5021348,  4.9489012],[11.936409,  11.155933,
10.270722,  9.6461763,  9.2041349,  8.8376112,  8.6274137,  8.4189205,
8.1638193,  7.9598513,  8.2124910,  8.1210442,  7.7346368,  7.1484294,
6.4850564]]
#Patrones de puntos en coordenadas X e Y de las diferentes galerías del
túnel propio 1.
#posicionPC_X=[[3.9925718,  3.6215830,  3.147297,  3.5517969,  3.5946498,
3.6353343,  3.6742172,  3.7827964,  4.0759149,  4.407239,  4.7791076,
5.2716184,  5.9447565,  6.8935294,  13.299760],[5.2122064,  5.2703290,
5.1523385,  5.3027821,  5.1489148,  5.2869635,  5.1245003,  5.2690721,
5.2612829,  5.1016788,  5.2473922,  4.9649005,  5.9284229,  6.8353977,
8.0126829,  9.6123772,  11.755064],[7.0364003,  6.3568754,  6.0596514,
6.1157560,  5.8404818,  5.9252868,  5.6732502,  5.7238150,  5.6981220,
5.4690752, 5.3566718, 6.3307405, 7.5132041, 8.7900724, 10.018202]]
#posicionPC_Y=[[11.643478, 10.225231, 7.5201802, 7.2344241, 6.9720588,
6.7229733,  6.4849052,  6.2915001,  6.1773500,  6.0504713,  5.9112434,
5.7674379,  5.6133313,  5.4338827,  5.6861157],[11.873248,  10.952395,
9.9208612,  9.3732328,  8.5385456,  8.1043081,  7.4283433,  7.0792475,
6.6331434,  6.1218543,  5.8373084,  5.3359795,  5.4458280,  5.4300065,
5.4094715,  5.3814969,  5.3043013],[11.976376,  10.562423,  9.6869173,
9.1950016,  8.5035076,  8.1148586,  7.5550375,  7.2007756,  6.8266125,
6.3971968, 6.0433302, 5.9841762, 5.8662481, 5.6411080, 5.2818689]]
#Patrones de puntos en coordenadas X e Y de las diferentes galerías del
túnel propio 2.
#posicionPC_X=[[3.9925718,  3.6215830,  3.147297,  3.5517969,  3.5946498,
3.6353343,  3.6742172,  3.7827964,  4.0759149,  4.407239,  4.7791076,
5.2716184,  5.9447565,  6.8935294,  13.299760],[6.0208173,  6.0623937,
5.9678869,  5.8853269,  5.3094034,  5.2495184,  5.0652142,  5.0256186,
5.0996699,  4.9425731,  4.9458323,  5.1132755,  7.0713778,  7.7951989,
9.8274784,  11.475937],[5.7961564,  5.8767400,  5.8604355,  5.7882624,
5.2131734,  5.1611972,  5.1141691,  4.9859438,  4.9519129,  4.9361358,
4.8933921,     4.9811525,     5.8229713,     7.5487628,     9.4932070,
10.976364],[5.7943273,  5.8749533,  5.8496132,  5.7775712,  5.3096342,
5.2566938,  5.1667976,  5.0786614,  5.1052999,  5.0870924,  4.9627361,
5.0107121, 5.5414681, 7.5476646, 9.4920311, 11.644480, 12.169874]]
#posicionPC_Y=[[11.643478, 10.225231, 7.5201802, 7.2344241, 6.9720588,
6.7229733,  6.4849052,  6.2915001,  6.1773500,  6.0504713,  5.9112434,
5.7674379,  5.6133313,  5.4338827,  5.6861157],[11.866556,  11.159534,
10.390018,  9.7177896,  8.7138853,  8.2262897,  7.6996517,  7.3103738,
7.0061903,  6.6071134,  6.3041849,  6.0664358,  6.3007064,  6.0624318,
5.9622574,  5.6115389],[11.986356,  11.308371,  10.598769,  9.9122715,
8.8674154,  8.3730383,  7.9257374,  7.4645529,  7.0965486,  6.7612133,
6.4327278,     6.1690087,     6.1299171,     6.2178402,     6.1652298,
5.8504319],[11.983962,  11.306267,  10.587320,  9.9020882,  8.9500761,
8.4465494,  7.9620194,  7.5215402,  7.1800752,  6.8334241,  6.4615335,
6.1795063, 6.0462861, 6.2175756, 6.1650109, 5.9387178, 5.3623667]]
pub = rospy.Publisher('informacionTunel', String)  #Se crea el objeto
para poder publicar mensajes en el topic informacionTunel.
valorGaleria=0 #Indica el patrón de puntos que se toma del túnel.
th_1=0 #Valores anteriores por si la transformada de Hough no funciona.
rh_1=0
siguienteGaleria=False #Variable que indica si se puede cambiar de patrón
de puntos para detectar otra galería.
galeriaADetectar=1 #Valor de la galería que se va a detectar.
soloUnaVez=True #Para hacer la figura una única vez cuando la posición
relativa es cero.
valorFigure=1 #Para crear diferentes figuras automáticamente.
```

```
def callback(msg):  #Función que se ejecuta cada vez que se recibe un mensaje del topic, en este caso /laser_front/scan, al que se está suscrito.
      global  Laser_X,  Pattern_X,  errores_min,  errores_pos,  deltax
      global Pattern_Y, Laser_Y
      global Metric, Method, posicionPC_X, posicionPC_Y, valorGaleria,
      global th_1, rh_1, siguienteGaleria
      global  galeriaADetectar,  soloUnaVez,  valorFigure  #Se  hacen variables globales para poder usar y reconocer sus  valores  en  las funciones que es encuentran en el archivo deftfg.py
      angles=np.arange(msg.angle_min,(msg.angle_max+msg.angle_incremen t),msg.angle_increment) #Se crea el vector de ángulos con  información del mensaje tipo LaserScan que se ha recibido.
      px=posicionPC_X[valorGaleria] #Se asignan los puntos del patrón en coordenadas X e Y para detectar la galería.
      py=posicionPC_Y[valorGaleria]
      r,  ang=deftfg.R_filter(angles,msg.ranges,30.0)  #Filtro  donde  se elimina todo punto que tenga una coordenada radial mayor a 30m.
      lx, ly=deftfg.Polar2Cart(ang,r) #Se pasa de coordenadas polares a cartesianas para realzar el emparejamiento con los puntos del patrón.
      I=[i for i,x in enumerate(ly) if x<=8.0] #Se guardan los índices de la posición de los puntos que cumplen la condición, en este caso se quieres eliminar los puntos con una coordenada Y mayor a 8m para que no afecten a las transformada de Hough.Realiza la misma  función  que  el find en MATLAB.
      lx_HT=deftfg.guardarEnLista(I,lx) #Se emplea la función definida en el archivo deftfg que guarda en un variable el contenido de otra.
      ly_HT=deftfg.guardarEnLista(I,ly)
      rho,theta=deftfg.houghTransform(lx_HT,ly_HT) #Se emplea la función definida  en el archivo deftfg que realiza la transformada de Hough.
      if theta>=-0.05 and theta<=0.05: #Si la transformada de Hough sale cero, es decir, errónea, se les asigna el valor anterior a las variables de salida de la transformada.
            theta=th_1
            rho=rh_1
      #Matriz de rotacion para el laser
      alfa=-((math.pi/2.0) + theta) #Ángulo que se rota en eje Z.
      R=np.array([[math.cos(alfa) , -math.sin(alfa)], [math.sin(alfa), math.cos(alfa)]]) #Matriz de  rotación en el eje Z 2x2.
      lr=np.array([lx, ly])
      #Se rota la traza del láser con la matriz de rotación.
      laser_rotado=np.dot(R,lr) #Multiplicación de matrices.
      laserFlip=np.fliplr(laser_rotado)  #Invierte  las  filas  por  las columnas y viceversa, para tener dos columnas y sacar las  coordenadas de los puntos de la traza del láser más fácilmente.
      lx=laserFlip[0,:]
      ly=laserFlip[1,:]
      lx, ly=deftfg.Y_filter(lx, ly, 0.0) #Se realiza otro filtro, en este caso se elimina todo punto que tiene una coordenada en Y negativa.
      ly=deftfg.sumaListayEscalar(ly,rho)
      #Se asigna  valor a las variables  globales  utilizadas  en  las funciones XPatternDistance y  PatternDistance.
      Pattern_X=px;      Pattern_Y=py;      Laser_X=lx;      Laser_Y=ly;
      Metric='Nearest'; Method='M'
      xmin=optimize.fminbound(deftfg.XPatternDistance,    -5.0,    5.0,
      args=(Pattern_X, Pattern_Y,  Laser_X,  Laser_Y,  Metric,  Method))
```

```
      #Se aplica la función XPatternDistance, cuya definición se
encuentra en el archivo deftfg, en un rango de -5m y 5m.
      err_min=deftfg.XPatternDistance(xmin,    Pattern_X,    Pattern_Y,
      Laser_X, Laser_Y, Metric, Method)  #Devuelve el error mínimo.
      err_pos=deftfg.XPatternDistance(0, Pattern_X, Pattern_Y, Laser_X,
      Laser_Y, Metric, Method) #Devuelve el error de posición respecto
al punto de referencia de la galería.
      if xmin<0.5 and xmin>-0.5 and err_min<0.5: #Conclusiones para
saber si la galería esta detecta.
            siguienteGaleria=True
            pub.publish("Galeria detectada") #Se publica que la galería
es detectada en el topic infomacionTunel.
            if xmin<0.05 and xmin>-0.05 and soloUnaVez==True: #Condición
para que se realice la figura justo en el punto de referencia de la
galería.
                  galeriaADetectar=galeriaADetectar+1  #Se detecta la
siguiente galería.
                  soloUnaVez=False #Para no volver a realizar la figura.
                  plt.ion()  #Se utiliza la librería matplotlib para
realizar figuras igual que en MATLAB, la función ion hace que se realicen
cambios automáticos en las gráficas.
                  plt.figure(valorFigure)#Se cambia el valor de la
figura para tener una por detección.
                  plt.plot(lx, ly, '.')
                  plt.title('LASER IN MOB_REF')
                  plt.hold(True) #Se pueden dibujar otros puntos en la
misma figura.
                  plt.plot(px, py, 'o')
                  plt.hold(False)
                  plt.show() #Para mostrar la figura.
                  plt.pause(0.1)  #Pausa para que le dé tiempo a
representar todo.
                  valorFigure=valorFigure+1

            else:
                  pub.publish("Galeria NO detectada") #Se publica en el
topic informacionTunel que la galería no es detectada en el topic.
      if xmin<=-3.5 and siguienteGaleria==True: #Condición para cambiar
los puntos del patrón a los de la siguiente galería.
            valorGaleria=valorGaleria+1  #Se realiza incrementando el
valor en uno y coger otros puntos de la lista.
            siguienteGaleria=False #Variables a condiciones iniciales.
            soloUnaVez=True
            del errores_min[:]
      if valorGaleria>=len(px): #Para que al final de detectar la última
galería no se produzca ningún error.
            valor=len(px)
            valorGaleria=valor-1
      errores_min.append(err_min) #Se van añadiendo a las listas los
valores.
      errores_pos.append(err_pos)
      deltax.append(xmin)
      errores_min_I=[]
      I=[i for i,x in enumerate(errores_min) if x<0.5]
      #Se guardan en una nueva lista los valores que tengan un error
mínimo de 0.5m.
      errores_min_I=deftfg.guardarEnLista(I,errores_min)
```

```
      #Se publica mediante un mensaje tipo String al topic
informacionTunel cuál es la última galería detectada.
      pub.publish("Ultima galeria detectada: galeria "+
      str(galeriaADetectar-1))
      #Se publica mediante un mensaje tipo String al topic
informacionTunel cuál es la siguiente galería a detectar.
      pub.publish("Galeria a detectar: galeria "+
      str(galeriaADetectar))
      #Se publica mediante un mensaje tipo String al topic
informacionTunel cuál es la distancia a la galería, es decir, la
posición relativa del robot frente a ella.
      pub.publish("La distancia a la galeria es "+ str(xmin))
      th_1=theta
      rh_1=rho

rospy.init_node('test_tfg') #Se inicia el nodo, el cual se llama
test_tfg, que contiene el siguiente archivo.
sub = rospy.Subscriber('/laser_front/scan', LaserScan, callback) #Se
subscribe al topic /laser_front/scan que nos proporciona mensajes
LaserScan y activa la función de llamada callback, cuyo primer parámetro
es el mensaje leído.
rospy.spin() #Para que el nodo no se apague hasta que es cerrado.
```

Deftfg.py

```
# -*- coding: utf-8 -*-
"""defTfg.ipynb
Automatically generated by Colaboratory. """
import numpy as np
import math
from scipy.spatial import cKDTree
#En este fichero se encuentran todas las definiciones de las funciones
creadas para implementar el método de detección de galerías en ROS.
def R_filter(tita, rang, rmax):
   #Esta función devuelve nuevas listas donde no se encuentran los
puntos que no cumplan la condición de rango máximo. El rango máximo
tendrá un valor de 30m.
   r=[]
   ang=[]
   datosOptimos=[i for i,x in enumerate(rang) if x<rmax] #Find en
MATLAB.
   for i in range(0,len(datosOptimos)):
       r.append(rang[datosOptimos[i]])
       ang.append(tita[datosOptimos[i]])
   return r, ang

def Y_filter(lx, ly, LACompara):
   #Esta función devuelve nuevas listas donde no se encuentran los
puntos que no cumplen la condición. #En este caso los que tienen una
coordenada en Y mayor a cero.
   lxf=[]
   lyf=[]
   datosOptimos=[i for i,x in enumerate(ly) if x>LACompara] #Find en
MATLAB.
   for i in range(0,len(datosOptimos)):
       lxf.append(lx[datosOptimos[i]])
```

```
            lyf.append(ly[datosOptimos[i]])
    return lxf, lyf

def Polar2Cart(ang, r):
    #Conversion de coordenadas polar a cartesianas. Devuelve dos nuevas
listas, una con las coordenadas en X y otra en Y.
    #X hacia delante del robot e Y hacia la izquierda del robot
    x_refRobot=[]
    y_refRobot=[]
    for i in range(0,len(r)):
        x_refRobot.append(r[i]*math.cos(ang[i]))
        y_refRobot.append(r[i]*math.sin(ang[i]))
    return x_refRobot, y_refRobot

def Cart2Polar(xp, yp):
    #Conversión de coordenadas cartesianas a polares. Devuelve dos
listas una con las coordenadas radiales y otra con las angulares.
    r=[]
    tita=[]
    datosOptimos=[i for i,x in enumerate(xp) if x>0]
    for i in range(0,len(datosOptimos)):
        r.append(math.sqrt(xp[datosOptimos[i]]**2 +
yp[datosOptimos[i]]**2)) #Operación para obtener la coordenada radial.
        ytita=yp[datosOptimos[i]]
        xtita=xp[datosOptimos[i]]
        tita.append(math.atan2(ytita,xtita)) #Operación para obtener la
coordenada angular.
    return r,tita

def PatternDistance(px, py, lx, ly, metric, method):
    #En esta función estas definidas todas las funciones de las
diferentes métricas y todos los métodos de cálculo del error de
posición.
    global d

    def polar():
        #La función que realiza la métrica Polar. Te devuelve la lista
de distancias.
        ldist,lang=Cart2Polar(lx,ly)
        pdist,pang=Cart2Polar(px,py)
        global d
        d=[]
        for i in range(0,len(pang)):
             ind=[j for j,x in enumerate(pang) if lang>x]
             if len(ind) ==0:
                 d.append(80.0-pdist[i])
             else:
                 d.append(abs(ldist[ind[0]] - pdist[i])
       return d

    def nearest():
       #La función que realiza la métrica Nearest. Te devuelve la lista
de distancias mínimas.
       global d
       d=[]
       A=np.zeros((len(lx),2)) #Se crean dos matrices de ceros.
       B=np.zeros((len(px),2))
```

```
      for i in range(0,len(lx)):
            A[i,0]=lx[i]
            A[i,1]=ly[i]
      for i in range(0,len(px)):
            B[i,0]=px[i]
            B[i,1]=py[i]
      tree = cKDTree(A) #Objeto de la clase cKDTree.
      d,k= tree.query(B) #Realiza la misma función que el dsearchn de
MATLAB, devolviéndote las distancias mínimas a cada punto del patrón.
      return d

   def x():
      #La función que realiza la métrica X. Te devuelve la lista de
distancias mínimas.
      global d
      d=[]
      py_np=np.array(py)
      ly_np=np.array(ly)
      restaList=[]
      for i in range(0,len(py)):
            resta=ly_np-py[i]
            for j in range(0,resta.size):
                  restaList.append(resta[j])
      aux=min(restaList)
      I = restaList.index(aux)
      del restaList[:]
      if aux <0.5:
            d.append(abs(px[i]-lx[I]))
      else:
            d.append(80.0)
      return d

   def q():
      #Función que devuelve el error cuadrático medio.
      global d
      d2=[]
      for i in range (0,len(d)):
            d2.append(d[i]**2) #Se elevan al cuadrado las diferentes
distancias de la lista.
      Distance= math.sqrt(sum(d2)/len(d))
      return Distance

   def m():
      #Función que devuelve el error medio.
      global d
      Distance= sum(d)/len(d)
      return Distance

   def h():
      #Función que devueve el error máximo.
      global d
      Distance= max(d)
      return Distance

   switcher_metric = { 'Polar': polar,  'Nearest': nearest, 'X': x }
#Diccionario de métricas.
```

```
   switcher_method = { 'Q': q, 'M': m, 'H': h } #Diccionario de
métodos de obtención del error de posición.
   d = switcher_metric.get(metric, lambda: "Invalid distance")
#Proporciona la lista de distancias resultantes de la métrica elegida.
 Distance = switcher_method.get(method, lambda: "Invalid method")
#Proporciona el error de posición o distancia del método   elegido.
 return Distance

def houghTransform (lx, ly):
   #Función en la que se realiza la transformada de Hough y se
calculan las coordenas radial y angular del robot respecto a los
muros.
   thetaSampleFrequency = math.pi/180
   rho=[]
   for i in np.arange(-30.0,30.2,0.2): #Se crea la lista de rangos.
      rho.append(round(i,2)) #Con dos decimales.
   theta=[]
   for i in np.arange(- math.pi , 0.0 + thetaSampleFrequency ,
thetaSampleFrequency): #Se crea la lista de ángulos.
       theta.append(i)
   numThetas=len(theta) #Longitud de la lista de titas.
   houghSpace =np.zeros((len(rho),numThetas)) #Matrices de ceros.
   accumulator =np.zeros((len(lx),numThetas))
   accumulatorX =np.zeros((len(lx),numThetas))
   accumulatorY =np.zeros((len(lx),numThetas))
   for i in range(len(lx)):
       for j in range(numThetas):
            accumulatorX[i,j]=lx[i]*math.cos(theta[j])
            accumulatorY[i,j]=ly[i]*math.sin(theta[j])
            #Se realiza cada coordenada por separado y se suman
posteriormente, porque si no se produce un error.
   accumulator = accumulatorX + accumulatorY
   for i in range(numThetas):
       hist =np.histogram(accumulator[:,i],rho) #Se realiza el
histograma para saber que coordenada radial se repite más.
   hist301=[]
   for z in range(hist.shape[0]):
       hist301.append(hist[z])
   hist301.append(0) #Se añade un valor extra al histograma porque
np.histogram lo realiza una columna de menos, ya que el último valor
de rho no se tiene en cuenta.
   houghSpace[:,i]=hist301 #Se aplica el histograma a la columna
correspondiente de la matriz houghSpace.
   del hist301[:] #Se elimina la información de la lista.
   I=[i for i,x in enumerate(rho) if x>=0]
   rhoNew=[]
   for i in range(0,len(I)): #Se eliminan todos los puntos con
coordenada radial negativa.
       rhoNew.append(rho[I[i]])
   houghSpaceNew=houghSpace[I,:] #Se realiza una nueva matriz, conlos
datos restantes.
   Y=houghSpaceNew.max(0) #Devuelve la columna con valor máximo.
   Ind=np.argmax(houghSpaceNew,0) #Devuelve la posición de la matriz
donde se encuentra el máximo.
   J=np.argmax(Y) #Devuelve la posición del valor máximo de la
columna.
   rhoF=rhoNew[Ind[J]] #Se guardan los valores finales.
```

```
    thetaF=theta[J]
    return rhoF, thetaF

def sumaListayEscalar(Lista, escalar):
    #Función para poder sumar un escalar a todos los valores de una
lista, ya que en la callback no se pueden realizar un bucle for.
    resultado=[]
    for i in range(0,len(Lista)):
        suma=Lista[i] + escalar
    resultado.append(suma)
    return resultado

def XPatternDistance(x, px, py, lx, ly, metric, method):
    #Función que va realizando la función PatternDistance, sumando un
valor que puede ser positivo o negativo a la coordenada X de los
puntos del patrón y devuelve el error de posición resultante.
    d=PatternDistance(sumaListayEscalar(px, x), py, lx, ly, metric,
method)
    return d

def guardarEnLista(Lista_pri, Lista_seg):
    #Función para poder guardar el contenido de una lista en otra, ya
que en la callback no se pueden realizar un bucle for.
    salida=[]
    for i in range(0,len(Lista_pri)):
        salida.append(Lista_seg[Lista_pri[i]])
    return salida
```

8.3 Archivos de los túneles sintéticos de Gazebo

Estos archivos se escriben en lenguaje de programación xml.

TunelPropio1.world

```
<?xml version="1.0" ?>
<sdf version="1.6">
  <world name="tunel_propio_1">
      <gui fullscreen='0'>
         <camera name='user_camera'>  <!--Posición de la cámara -->
            <pose>-6.3 -4.2 3.6 0 0.268 0.304</pose>
         </camera>
      </gui>
      <scene>
         <ambient>0.2 0.2 0.2 1.0</ambient>
         <background>0.34 0.39 0.43 1.0</background>
         <grid>false</grid>
         <origin_visual>false</origin_visual>
      </scene>
      <light name='user_point_light_0' type='point'> <!--Luces para
poder ver en el interior del tunel -->
         <pose>30.0 0.0 3.0 0 -0 0</pose>
         <diffuse>0.5 0.5 0.5 1</diffuse>
         <specular>0.1 0.1 0.1 1</specular>
         <attenuation>
            <range>40</range>
            <constant>0.05</constant>
```

```
            <linear>0.0</linear>
            <quadratic>0.0</quadratic>
         </attenuation>
         <cast_shadows>0</cast_shadows>
         <direction>0 0 -1</direction> <--Para que la luz sea hacia el
suelo, dirección contraria a la del eje Z-->
      </light>
      <light name='user_point_light_1' type='point'>
         <pose>60.0 0.0 3.0 0 -0 0</pose>
         <diffuse>0.5 0.5 0.5 1</diffuse>
         <specular>0.1 0.1 0.1 1</specular>
         <attenuation>
            <range>40</range>
            <constant>0.05</constant>
            <linear>0.0</linear>
            <quadratic>0.0</quadratic>
         </attenuation>
         <cast_shadows>0</cast_shadows>
         <direction>0 0 -1</direction>
      </light>
      <!—Secciones del tunel -->
      <include>
         <uri>model://tunnel_entrance</uri> <!--Se incluye la entrada
del túnel con su posición-->
         <name>tile_0</name>
         <--Posición en X, Y y Z y giro respecto a X, Y y Z-->
         <pose>10 0 0 0 0 -1.5708</pose>
      </include>
      <include>
         <uri>model://tunnel_straight</uri> <!--El modelo y cada una
de las posiciones de las secciones que contiene el túnel -->
         <name>tile_1</name>
         <pose>15 0 0 0 0 1.5708</pose>
      </include>
      <include>
         <uri>model://tunnel_straight</uri>
         <name>tile_2</name>
         <pose>20 0 0 0 0 1.5708</pose>
      </include>
      <include>
         <uri>model://tunnel_intersection</uri>
         <name>tile_3</name>
         <pose>30 0 0 0 0 1.5708</pose>
      </include>
      <include>
         <uri>model://tunnel_tile_5</uri>
         <name>tile_4</name>
         <pose>22.5 17 0 0 0 0</pose>
      </include>
      <include>
         <uri>model://tunnel_straight</uri>
         <name>tile_5</name>
         <pose>30 0 0 0 0 1.5708</pose>
      </include>
      <include>
         <uri>model://tunnel_intersection_t</uri>
         <name>tile_6</name>
```

```
        <pose>36 7.5 0 0 0 3.1416</pose>
      </include>
      <include>
        <uri>model://tunnel_corner_left</uri>
        <name>tile_7</name>
        <pose>36 3.2 0 0 0 0</pose>
      </include>
      <include>
        <uri>model://tunnel_straight</uri>
        <name>tile_8</name>
        <pose>48 0 0 0 0 1.5708</pose>
      </include>
      <include>
        <uri>model://tunnel_intersection</uri>
        <name>tile_9</name>
        <pose>60 0 0 0 0 1.5708</pose>
      </include>
      <include>
        <uri>model://tunnel_bend_right</uri>
        <name>tile_10</name>
        <pose>52.5 2.9 0 0 0 0</pose>
      </include>
      <include>
        <uri>model://tunnel_tile_5</uri>
        <name>tile_11</name>
        <pose>65 0 0 0 0 1.5708</pose>
      </include>
      <include>
        <uri>model://tunnel_tile_5</uri>
        <name>tile_12</name>
        <pose>22.5 31 0 0 0 0</pose>
      </include>

      <!--Bloques que se han incluido en el túnel para cerrar el final
del túnel o las intersecciones -->
      <include>
        <uri>model://tunnel_tile_blocker</uri>
        <name>blocker_1</name>
        <pose>26.7 5.45 0 0 0 0</pose>
      </include>
      <include>
        <uri>model://tunnel_tile_blocker</uri>
        <name>blocker_2</name>
        <pose>22.75 -7.76 0 0 0 0</pose>
      </include>
      <include>
        <uri>model://tunnel_tile_blocker</uri>
        <name>blocker_3</name>
        <pose>52.725 -7.72 0 0 0 0</pose>
      </include>
      <include>
        <uri>model://tunnel_tile_blocker</uri>
        <name>blocker_4</name>
        <pose>70 0.295 0 0 0 0</pose>
      </include>
  </world>
</sdf>
```

TunelPropio2.world

```
<?xml version="1.0" ?>
<sdf version="1.6">
  <world name="tunnel_propio_2">
  <!--Este archivo cambia respecto al otro en que se han introducido
secciones diferentes para tener un túnel distinto al otro -->
    <gui fullscreen='0'>
      <camera name='user_camera'>
        <pose>-6.3 -4.2 3.6 0 0.268 0.304</pose>
      </camera>
    </gui>
    <scene>
      <ambient>0.2 0.2 0.2 1.0</ambient>
      <background>0.34 0.39 0.43 1.0</background>
      <grid>false</grid>
      <origin_visual>false</origin_visual>
    </scene>
    <light name='user_point_light_0' type='point'>
      <pose>30.0 0.0 3.0 0 -0 0</pose>
      <diffuse>0.5 0.5 0.5 1</diffuse>
      <specular>0.1 0.1 0.1 1</specular>
      <attenuation>
        <range>40</range>
        <constant>0.05</constant>
        <linear>0.0</linear>
        <quadratic>0.0</quadratic>
      </attenuation>
      <cast_shadows>0</cast_shadows>
      <direction>0 0 -1</direction>
    </light>
    <light name='user_point_light_1' type='point'>
      <pose>60.0 0.0 3.0 0 -0 0</pose>
      <diffuse>0.5 0.5 0.5 1</diffuse>
      <specular>0.1 0.1 0.1 1</specular>
      <attenuation>
        <range>40</range>
        <constant>0.05</constant>
        <linear>0.0</linear>
        <quadratic>0.0</quadratic>
      </attenuation>
      <cast_shadows>0</cast_shadows>
      <direction>0 0 -1</direction>
    </light>
    <!--Secciones del túnel -->
    <include>
      <uri>model://tunnel_entrance</uri>
      <name>tile_0</name>
      <pose>10 0 0 0 0 -1.5708</pose>
    </include>
    <include>
      <uri>model://tunnel_straight</uri>
      <name>tile_1</name>
      <pose>15 0 0 0 0 1.5708</pose>
    </include>
    <include>
      <uri>model://tunnel_intersection</uri>
      <name>tile_2</name>
      <pose>30 0 0 0 0 1.5708</pose>
```

```
</include>
<include>
  <uri>model://tunnel_intersection</uri>
  <name>tile_3</name>
  <pose>41.79 0 0 0 0 1.5708</pose>
</include>
<include>
  <uri>model://tunnel_intersection</uri>
  <name>tile_4</name>
  <pose>53.358 0 0 0 0 1.5708</pose>
</include>
<include>
  <uri>model://tunnel_intersection</uri>
  <name>tile_5</name>
  <pose>64.45 0 0 0 0 1.5708</pose>
</include>
<include>
  <uri>model://tunnel_tile_5</uri>
  <name>tile_6</name>
  <pose>22.5 17 0 0 0 0</pose>
</include>
<include>
  <uri>model://tunnel_tile_5</uri>
  <name>tile_7</name>
  <pose>22.5 31 0 0 0 0</pose>
</include>
<include>
  <uri>model://tunnel_tile_5</uri>
  <name>tile_8</name>
  <pose>34.3 17 0 0 0 0</pose>
</include>
<include>
  <uri>model://tunnel_tile_5</uri>
  <name>tile_9</name>
  <pose>34.3 31 0 0 0 0</pose>
</include>
<include>
  <uri>model://tunnel_tile_5</uri>
  <name>tile_10</name>
  <pose>45.8 17 0 0 0 0</pose>
</include>
<include>
  <uri>model://tunnel_tile_5</uri>
  <name>tile_11</name>
  <pose>45.8 31 0 0 0 0</pose>
</include>
<include>
  <uri>model://tunnel_tile_5</uri>
  <name>tile_12</name>
  <pose>56.9 17 0 0 0 0</pose>
</include>
<include>
  <uri>model://tunnel_tile_5</uri>
  <name>tile_13</name>
  <pose>56.9 31 0 0 0 0</pose>
</include>
<include>
  <uri>model://tunnel_straight</uri>
  <name>tile_14</name>
  <pose>69 0 0 0 0 1.5708</pose>
</include>
```

```
    <!--Bloques que se han incluido en el túnel para cerrar el final
del túnel o las intersecciones o cambiar la esquina de las galerías y
el láser las vea distintas-->
    <include>
      <uri>model://tunnel_tile_blocker</uri>
      <name>blocker_1</name>
      <pose>26.7 5.45 0 0 0 0</pose>
    </include>
    <include>
      <uri>model://tunnel_tile_blocker</uri>
      <name>blocker_2</name>
      <pose>22.75 -7.76 0 0 0 0</pose>
    </include>
    <include>
      <uri>model://tunnel_tile_blocker</uri>
      <name>blocker_3</name>
      <pose>34.48 -7.76 0 0 0 0</pose>
    </include>
    <include>
      <uri>model://tunnel_tile_blocker</uri>
      <name>blocker_4</name>
      <pose>70 0.29 0 0 0 0</pose>
    </include>
    <include>
      <uri>model://tunnel_tile_blocker</uri>
      <name>blocker_5</name>
      <pose>46.12 -7.76 0 0 0 0</pose>
    </include>
    <include>
      <uri>model://tunnel_tile_blocker</uri>
      <name>blocker_6</name>
      <pose>57.23 -7.76 0 0 0 0</pose>
    </include>
    <include>
      <uri>model://blq1</uri> <!--Bloque 1 de creación propia -->
      <name>personal_blocker_1</name>
      <pose>37 3.42 0.5 0 0 0</pose>
    </include>
    <include>
      <uri>model://blq2</uri> <!--Bloque 2 de creación propia -->
      <name>personal_blocker_2</name>
      <pose>48.3 3 0.5 0 0 0</pose>
    </include>
    <include>
      <uri>model://blq3</uri> <!--Bloque 3 de creación propia -->
      <name>personal_blocker_3</name>
      <pose>59.82 3.966 -0.5 0 0 0</pose>
    </include>
  </world>
</sdf>
```